

Mikrokomputery

Jan Bielecki

TURBO PASCAL z grafiką dla IBM PC

TURBO PASCAL
z grafiką dla IBM PC

Firmie Borland International
w uznaniu jej wkładu w rozwój
mikroinformatyki

Mikrokomputery

Jan Bielecki

TURBO PASCAL z grafiką dla IBM PC



Wydawnictwa Naukowo-Techniczne
Warszawa 1989

Opiniodawca *Jędrzej Wróblewski*
Redaktor *Ewa Zdanowicz*
Redaktor techniczny *Krystyna Orloś*
Projektant serii *Juliusz Rybicki*
Okładkę i strony tytułowe projektował *Andrzej Pilich*

681.3.06

W książce przedstawiono język programowania mikrokomputerów — Turbo Pascal w jego wersji dla mikrokomputera IBM PC. Dokładnie omówiono wszystkie konstrukcje języka i zestawy podprogramów standardowych, opisy słowne uzupełniając licznymi i szczegółowo skomentowanymi przykładami programów. Wiele miejsca poświęcono problematyce programowania operacji wejścia/wyjścia oraz zastosowaniom języka jako narzędzia do programowania zagadnień z zakresu grafiki komputerowej na mikrokomputerach rodziny IBM PC. Książka jest przeznaczona dla szerokiego kręgu Czytelników — użytkowników popularnych mikrokomputerów.

ISBN 83-204-0944-6

Spis treści

Przedmowa

Język Turbo Pascal jest najbardziej rozpowszechnionym językiem programowania mikrokomputerów. Został on opracowany przez firmę Borland International jako dialekt języka wzorcowego Pascal, ale dzięki swojej popularności sam stał się wzorcem języka programowania mikrokomputerów zarówno 8-, jak i 16-bitowych.

Do najważniejszych zalet Turbo Pascala należy zaliczyć: mały rozmiar kompilatora, znaczną zgodność języka z Pascalem wzorcowym, bardzo szybką kompilację programów, scalenie kompilatora z interakcyjnym edytorem ekranowym, sygnalizowanie błędów na poziomie programu źródłowego, obszerną bibliotekę podprogramów i przydatne rozszerzenia ułatwiające programowanie systemowe.

Niniejsze opracowanie zawiera szczegółową prezentację języka Turbo Pascal dla mikrokomputerów IBM PC. Dotyczy to jednak głównie rozdziałów poświęconych grafice, ponieważ pozostała część książki stosuje się również do innych mikrokomputerów, takich jak np. Amstrad/Schneider 6128, Elwro 800 Jr oraz wszystkich zgodnych z IBM PC.

Jan Bielecki

1. Wstęp

Nazwa Turbo Pascal dotyczy interakcyjnego systemu programowania, składającego się z kompilatora języka Turbo Pascal oraz zintegrowanego z nim edytora ekranowego.

Edytor systemu Turbo Pascal jest wzorowany na powszechnie znanym edytorze WordStar, a interakcyjność systemu przejawia się w głównej mierze wygodą edycji oraz sygnalizowaniem błędów wykrytych w programach bezpośrednio w obrębie tekstu źródłowego.

Naturalne i wymagające minimalnej liczby manipulacji przechodzenie między edycją, kompilacją i wykonaniem programu zapewnia, że uruchamianie programów w systemie Turbo Pascal odbywa się prawie wyłącznie na poziomie źródłowym. Znaczna szybkość kompilowania programów powoduje natomiast, że przejście od programu źródłowego do programu wykonywalnego jest niemal natychmiastowe. Wydatnie skraca to cykl modyfikowania programu i przyśpiesza uzyskanie jego wersji końcowej.

W celu zilustrowania zasad posługiwania się systemem Turbo Pascal zostanie rozpatrzony prosty program do wyznaczania objętości kuli o zadanym promieniu.

```
program Volume;
var
  Rad, Vol : real;
begin
  ClrScr;
  GotoXY(10, 11);
  Write('radius=');
  Readln(Rad);
  Vol := 4/3*Pi*Rad*Rad*Rad;
```

```
    GotoXY(10, 13);
    Writeln('volume= ', Vol);
    repeat until KeyPressed
  end.
```

W programie tym występuje predefiniowana nazwa *Pi* reprezentująca przybliżenie liczby π oraz nazwy symboliczne *Rad* i *Vol* reprezentujące odpowiednio promień i objętość kuli. Wykonanie procedury *ClrScr* powoduje wyczyszczenie ekranu składającego się z 80 kolumn i 25 wierszy, a wykonanie procedury *GotoXY* powoduje ustawienie kursora w pozycji o podanych współrzędnych. W przypadku wywołania *GotoXY*(10, 13) jest to 10 kolumna i 13 wiersz. Zarówno numery kolumn, jak i wierszy są liczone od 1 do wartości maksymalnej. Wykonanie cyklu **repeat until** powoduje wstrzymanie wykonywania programu do momentu wprowadzenia dowolnego znaku z klawiatury. Znak ten pozostaje w buforze i może być następnie zinterpretowany przez system Turbo Pascal.

W celu skompilowania i wykonania przytoczonego programu należy w pierwszej kolejności wywołać system Turbo Pascal. Odbywa się to za pomocą wywołania programu Turbo.

Bezpośrednio po wywołaniu tego programu na ekranie monitora pojawia się komunikat identyfikujący system

```
Turbo Pascal system
Copyright (C) ... by Borland Inc.
Include error messages (Y/N)?
```

W przytoczonym napisie jest zawarte pytanie, czy do systemu należy dołączyć zbiór tekstów stanowiących komunikaty. Ponieważ komunikaty te zajmują zaledwie 1,5 Kb pamięci, a ich występowanie znacznie ułatwia diagnostykę, zaleca się udzielenie odpowiedzi Y.

W chwilę po udzieleniu odpowiedzi na ekranie monitora pojawia się menu, w którym wybrane litery poszczególnych słów są wyróżnione przez rozjaśnienie.

```
Logged drive:
Active directory:
Work file:
Main file:
Edit  Compile Run  Save
Dir   Quit compiler Options
Text:
Free:
```

Wprowadzanie z klawiatury dowolnej dyrektywy mającej postać jednej z wyróżnionych liter (bez naciskania klawisza Enter) powoduje stworzenie warunków do interakcyjnej zmiany parametrów systemu.

Dyrektywa L
Wprowadzenie z klawiatury litery L umożliwia zmianę domniemanej stacji dyskowej, którą bezpośrednio po inicjacji jest stacja domniemana w systemie operacyjnym. Jeśli jest np. pożądane wyróżnienie stacji B, to bezpośrednio po zapytaniu

New drive:

należy wprowadzić z klawiatury parę znaków B: zakończoną znakiem Enter.

Dyrektywa A
Wprowadzenie z klawiatury litery A umożliwia zmianę domniemanego katalogu stacji dyskowej, którym bezpośrednio po inicjacji systemu jest katalog\. Jeśli jest np. pożądane wyróżnienie katalogu\JB\JBTB, to bezpośrednio po zapytaniu

New directory:

należy wprowadzić z klawiatury tekst\JB\JBTB zakończony znakiem Enter. Niezwłocznie po wykonaniu tej czynności nowa nazwa katalogu pojawi się w menu systemu.

Dyrektywa W
Wprowadzenie z klawiatury dyrektywy W umożliwia określenie nazwy zbioru roboczego zawierającego program źródłowy. Zbiór ten może już istnieć, ale również może być zbiorem, który ma być dopiero utworzony. Bezpośrednio po zapytaniu

Work file name:

należy wprowadzić tekst określający nazwę zbioru, zakończony znakiem Enter. Dla rozpatrywanego programu przykładowego mógłby to być np. tekst TB01.PAS.

W systemie CP/M i DOS nazwa zbioru może być określona jako dwuczłonowa. Pierwszy człon składa się z nie więcej niż 8 znaków, a drugi — nazywany *rozszerzeniem* — z nie więcej niż 4 znaków, z których pierwszy jest znakiem . (kropka). Jeśli w odpowiedzi na rozpatrywane zapytanie o nazwę zbioru ograniczyć się do pierwszego członu nazwy, to zostanie domniemana kropka oraz rozszerzenie .PAS. Aby uniknąć domniemania rozszerzenia, można ograniczyć się do podania pierwszego członu nazwy oraz kropki.

Należy nadmienić, że jeśli ustalenie nowej nazwy zbioru roboczego następuje w sytuacji, gdy nie zapamiętano na dysku poprzedniego zbioru roboczego, to

system wyprowadza komunikat zapytujący, czy zbiór poprzedni powinien zastać zapamiętany. Odpowiedzią na takie zapytanie jest zwykle Y (*yes*) albo N (*no*).

Dyrektywa M
Wprowadzenie z klawiatury dyrektywy M umożliwia określenie nazwy zbioru poddawanego kompilacji. Jeśli nazwa taka nie zostanie ustalona, to kompilacji podlega aktualny zbiór roboczy.

Zasady określania nazwy zbioru poddawanego kompilacji, nazywanego dalej *zbiorem głównym*, są takie same jak dla zbioru roboczego. Użycie zbioru głównego jest wygodne wówczas, gdy program źródłowy zawiera dyrektywy włączające w miejscu ich wystąpienia zawartość innych zbiorów. W takim przypadku, stwierdzenie błędu kompilacji w jednym ze zbiorów włączonych, czyni go zbiorem roboczym, nadającym się do natychmiastowej edycji. Po jej wykonaniu można w prosty sposób ponowić kompilację programu zawartego w zbiorze głównym.

Dyrektywa E
Wprowadzenie z klawiatury dyrektywy E powoduje wywołanie edytora i podjęcie edycji zbioru roboczego. Jeśli do tego momentu nie została określona nazwa zbioru roboczego, to system wyprowadza komunikat jak podczas wprowadzenia dyrektywy W, a edycję podejmuje dopiero po zidentyfikowaniu zbioru.

Dyrektywa C
Wprowadzenie z klawiatury dyrektywy C powoduje wykonanie kompilacji zbioru głównego. Jeśli do tego momentu nie określono nazwy tego zbioru, to podejmowana jest kompilacja zbioru roboczego. Jeśli określono nazwę zbioru głównego, a poddawano edycji zbiór roboczy, to rozpoczęcie kompilacji zostanie poprzedzone zapamiętaniem zbioru roboczego w pamięci zewnętrznej.

W następstwie wykonania kompilacji powstaje program rezydujący w pamięci operacyjnej albo program umieszczony w zbiorze z rozszerzeniem .COM albo .CHN. Decyzja o sposobie przeprowadzenia kompilacji jest podejmowana na podstawie opcji kompilacji ustawianych za pomocą dyrektywy O. Przez domniemanie przyjmuje się, że program wynikowy ma być umieszczony w pamięci operacyjnej.

Jeśli podczas wykonywania kompilacji zostanie wprowadzony z klawiatury dowolny znak, to kompilacja zostanie wstrzymana. Bezpośrednio po udzieleniu odpowiedzi na zapytanie

*** Abort compilation (Y/N)?

nastąpi zaniechanie albo kontynuowanie kompilacji.

Dyrektywa R
Wprowadzenie z klawiatury dyrektywy R powoduje wykonanie właśnie skompilowanego programu rezydującego w pamięci operacyjnej. Jeśli za pomocą dyrektywy O zostanie ustawiona opcja Com-file, to wykonanie będzie dotyczyć tego programu znajdującego się w pamięci zewnętrznej. Jeśli przed użyciem dyrektywy R nie posłużono się dyrektywą C, to podjęcie wykonywania programu zostanie automatycznie poprzedzone jego kompilacją.

Dyrektywa S
Wprowadzenie z klawiatury dyrektywy S powoduje zapamiętanie zbioru roboczego w pamięci zewnętrznej. Poprzednia wersja tego zbioru zostanie przemianowana w taki sposób, że otrzyma rozszerzenie .BAK.

Dyrektywa D
Wprowadzenie z klawiatury dyrektywy D umożliwia wyprowadzenie nazw zbiorów znajdujących się w katalogu. Bezpośrednio po zapytaniu

Dir mask:

należy podać nazwę zbioru albo nazwę rodziny zbiorów, np. B: J*B? (* oznacza dowolny dopuszczalny ciąg znaków, a ? — dowolny dopuszczalny znak). Jeśli udzielając odpowiedzi na zapytanie nie podano nazwy stacji dyskowej albo nazwy katalogu, to zostaną one domniemane zgodnie z menu.

Dyrektywa Q
Wprowadzenie z klawiatury dyrektywy Q powoduje zakończenie współpracy z systemem Turbo Pascal i wywołanie systemu operacyjnego. Jeśli przed użyciem dyrektywy Q poddawano edycji zbiór roboczy, to wywołanie systemu operacyjnego zostanie poprzedzone zapytaniem, czy zbioru tego nie należy zapamiętać w pamięci zewnętrznej.

Dyrektywa O
Wprowadzenie z klawiatury dyrektywy O umożliwia określenie opcji kompilacji. Przez domniemanie przyjmuje się, że program wykonywalny zostanie umieszczony w pamięci operacyjnej. Jeśli po użyciu dyrektywy O zostanie użyta poddyrektywa C, to program ten będzie umieszczony w pamięci zewnętrznej jako zbiór z rozszerzeniem .COM. Przywrócenie pierwotnego domniemania można uzyskać za pomocą poddyrektywy M.

Jeśli zostanie użyta poddyrektywa H, to program wykonywalny zostanie umieszczony w pamięci zewnętrznej jako zbiór o nazwie z rozszerzeniem .CHN. Program zawarty w takim zbiorze różni się tym od programu zawartego w zbiorze o nazwie z rozszerzeniem .COM, że nie zawiera biblioteki podprogramów standardowych i musi być wywoływany z innego programu za pomocą specjalnej procedury *Chain*.

Posłużenie się poddyrektywą P umożliwia określenie parametrów programu wykonywalnego umieszczonego w pamięci operacyjnej. Zostaną one przekazane programowi dokładnie w taki sposób, w jaki są przekazywane parametry programom wywoływanym z pamięci zewnętrznej.

Użycie poddyrektywy F umożliwia zlokalizowanie błędu wykonywania programu umieszczonego w zbiorze o nazwie z rozszerzeniem .COM albo .CHN. W odróżnieniu od programów wykonywalnych umieszczonych w pamięci operacyjnej, których błędy wykonywania są sygnalizowane przez wskazanie miejsca błędu w tekście źródłowym, błędy wykonywania programów umieszczonych w pamięci zewnętrznej są sygnalizowane w sposób zakodowany, przez podanie numeru błędu i stanu licznika instrukcji. Jeśli po zapytaniu wywołanym użyciem poddyrektywy F zostanie podany ten właśnie stan licznika instrukcji, to automatycznie zostanie wskazane miejsce w programie źródłowym, gdzie wystąpił błąd wykonywania.

Ostatnią poddyrektywą dyrektywy O jest poddyrektywa Q. Jej wykonanie kończy wykonywanie dyrektywy O i powoduje przywrócenie głównego menu.

Jak wynika z przytoczonego opisu, posługiwanie się dyrektywami i poddyrektywami systemu Turbo Pascal jest w miarę nieskomplikowane. Równie proste jest wprowadzanie i poprawianie programu źródłowego, następujące po użyciu dyrektywy E albo automatycznie, po zlokalizowaniu błędu podczas kompilowania programu.

Zasady posługiwania się edytorem systemu Turbo Pascal wynikają z jego podobieństwa do procesora tekstów WordStar, na którym jest on wzorowany. Odsyłając do dodatku wszystkich zainteresowanych pełnymi możliwościami tego edytora, wystarczy ograniczyć się do przytoczenia minimalnego zestawu dyrektyw umożliwiających obróbkę programów źródłowych.

Na wstępie należy zauważyć, że jeśli ktoś nie popełnia błędów, to bezpośrednio po wywołaniu edytora za pomocą dyrektywy E może wprowadzić tekst programu jako ciąg wierszy zakończonych znakami Enter, a następnie posłużyć się dyrektywą ^KD kończącą edycję i przywracającą główne menu systemu.

Dyrektywa KD składa się z liter K i D i podobnie jak pozostałe dyrektywy edytora systemu Turbo Pascal wymaga posłużenia się klawiszem Ctrl. Klawisz ten powinien być wciśnięty podczas wprowadzania liter K i D. Będzie to oznaczane skrótowo za pomocą opisu Ctrl-K-D albo ^KD.

W obrębie tekstu wyświetlanego na ekranie monitora jeden ze znaków jest wyróżniony za pomocą kursora. Operacje na tekście są zawsze wykonywane w odniesieniu do tego znaku, zawierającego go słowa albo wiersza.

Przesunięcie kursora o jedną pozycję w lewo, w prawo, w górę lub w dół odbywa się za pomocą dyrektyw Ctrl-S, Ctrl-D, Ctrl-E lub Ctrl-X. Klawisze S, D, E i X tych dyrektyw tworzą romb

```
  E
S   D
  X
```

a jak łatwo zapamiętać, dyrektywa Ctrl-E przesuwa kursor o jeden wiersz w górę, dyrektywa Ctrl-X przesuwa kursor o jeden wiersz w dół, dyrektywa Ctrl-S przesuwa kursor o jedną pozycję w lewo, a dyrektywa Ctrl-D przesuwa kursor o jedną pozycję w prawo. Dzięki takiemu rozwiązaniu o funkcji dyrektywy decyduje nie mnemonika nazwy jej klawisza, lecz położenie klawisza w przytoczonym układzie czterokierunkowym.

Usuwanie znaków tekstu odbywa się za pomocą klawisza ← (Backspace). Każdorazowe naciśnięcie tego klawisza powoduje usunięcie znaku znajdującego się z lewej strony znaku wyróżnionego przez kursor. Wstawianie nowych znaków między słowami lub w obrębie słów tekstu nie wymaga specjalnych zabiegów. Każdy z takich znaków jest wstawiany bezpośrednio przed znakiem wyróżnionym przez kursor, a pozostałe znaki wiersza zostają przesunięte o jedną pozycję w prawo. Usuwanie całych wierszy odbywa się za pomocą dyrektywy Ctrl-Y, usuwanie słów — za pomocą dyrektywy Ctrl-T, a wstawianie nowych wierszy — za pomocą dyrektywy Ctrl-N. Wiele innych dyrektyw i możliwości edytora opisano w dodatku D.

Podsumowując to, co przedstawiono w niniejszym rozdziale, można podać, że utworzenie i uaktywnienie programu przedstawionego na wstępie wymaga wykonania następujących czynności:

- Wywołania systemu Turbo Pascal
  TURBO
- Dołączenia do kompilatora zbioru komunikatów
  Y
- Określenia nazwy zbioru roboczego
  W
- Zainicjowania wprowadzenia programu źródłowego
  E
- Zakończenia edycji
  ^KD
- Polecenia wykonania kompilacji
  C
- Polecenia wykonania programu
  R

Po wykonaniu tych czynności można wywołać system operacyjny, posługując się w tym celu dyrektywą Q.

W przypadku gdyby było pożądane przechowanie programu wykonywalnego jako zbioru z rozszerzeniem .COM, należałoby przed wykonaniem kompilacji posłużyć się dyrektywą O, a następnie poddyrektywami C i Q. Po wykonaniu kompilacji i powrocie do systemu operacyjnego wspomniany program mógłby być wykonywany już bez udziału systemu Turbo Pascal.

2. Jednostki leksykalne, odstępy, komentarze

Przystępując do szczegółowego omówienia języka programowania Turbo Pascal, należy zacząć od jego *jednostek leksykalnych*, którymi — jak w każdym języku programowania — są *słowa kluczowe*, *identyfikatory*, *literały* i *ograniczniki*. Spacje, znaki tabulacji, znaki przejścia do nowego wiersza i komentarze nie stanowią jednostek leksykalnych. Każdy ciąg takich znaków i komentarzy jest traktowany tak jak pojedyncza spacja i będzie krótko nazywany *odstępem*. Użycie odstępu jest niezbędne jedynie wtedy, kiedy w programie źródłowym sąsiadują ze sobą identyfikatory albo słowa kluczowe.

Zgodnie z przytoczoną klasyfikacją każdy program zapisany w języku Turbo Pascal składa się z jednostek leksykalnych i odstępów. Wyodrębnienie tych obiektów odbywa się podczas analizowania programu, dokonywanego w naturalnym porządku, tj. od lewej do prawej i od góry do dołu. Za jednostkę leksykalną jest uznawany w takim przypadku najdłuższy ciąg znaków nie zawierający odstępów, który może uchodzić za jednostkę leksykalną.

Jednostki leksykalne i komentarze są tworzone ze *znaków* i *dwuznaków podstawowych*, do których zalicza się

- małe i duże litery alfabetu angielskiego, wraz ze znakiem podkreślenia
  _, *a*, *b*, *c*, *d*, *e*, *f*, *g*, *h*, *i*, *j*, *k*, *l*, *m*,
  n, *o*, *p*, *q*, *r*, *s*, *t*, *u*, *v*, *w*, *x*, *y*, *z*,
  A, *B*, *C*, *D*, *E*, *F*, *G*, *H*, *I*, *J*, *K*, *L*, *M*,
  N, *O*, *P*, *Q*, *R*, *S*, *T*, *U*, *V*, *W*, *X*, *Y*, *Z*
- cyfry dziesiętne
  0, 1, 2, 3, 4, 5, 6, 7, 8, 9
- znaki specjalne
  \+ − * / = { }
  ( ) [ ] < > ^
  . , : ; ' # $

- dwuznaki

  przypisania: :=
  relacje: <> <= >=
  zakresy: ..
  nawiasy: (* *) (. .)
  (nawiasy z gwiazdkami pełnią rolę nawiasów klamrowych, a nawiasy z kropkami pełnią rolę nawiasów kwadratowych).

Litery duże i małe nie są odróżniane. Powoduje to m.in., że identyfikatory takie jak np. *filename* i *FileName* są uznawane za identyczne.

Słowa kluczowe

Słowem kluczowym jest spójny ciąg liter tworzący jedno z wymienionych niżej słów

absolute	**external**	**nil**	**shl**
and	**file**	**not**	**shr**
array	**for**	**of**	**string**
begin	**forward**	**or**	**then**
case	**function**	**overlay**	**to**
const	**goto**	**packed**	**type**
div	**if**	**procedure**	**until**
do	**in**	**program**	**var**
downto	**inline**	**record**	**while**
else	**label**	**repeat**	**with**
end	**mod**	**set**	**xor**

Przykład
Najkrótszy program w języku Turbo Pascal.

```
begin
end.
```

- Program składa się z dwóch słów kluczowych i ogranicznika.
- Wykonanie tego programu nie wywołuje żadnych skutków. □

Identyfikatory

Identyfikatorem jest ciąg literowo-cyfrowy, rozpoczynający się od litery, taki który nie jest słowem kluczowym. „Literą" użytą w identyfikatorze może być dowolna mała albo duża litera alfabetu angielskiego oraz znak podkreślenia. Liczba znaków identyfikatora nie może przekroczyć 127. Wszystkie jego znaki są istotne.

Wiele identyfikatorów ma predefiniowane znaczenie i służy do oznaczania literałów, funkcji i procedur. Do tej grupy należą następujące identyfikatory:

Abs
Addr
Append
Arc
ArcTan
Assign
Aux
AuxInPtr
AuxOutPtr
Back
Bdos
Bios
BiosHL
Black
Blink
BlockRead
BlockWrite
Blue
Boolean
Brown
BufLen
BW40
BW80
Byte
C40
C80
Chain
Char
ChDir
Chr
Circle
Close
ClrEol
ClrScr
ClearScreen
ColorTable
Con
Concat
ConInPtr
ConOutPtr
ConstPtr
Copy
Cos
CrtExit
CrtInit
Cyan
DarkGray
DelLine
Delay
Delete
Dispose
Draw
Eof
Eoln
Erase
Execute
Exit
Exp
False
FilePos
FileSize
FillChar
FillPattern
FillScreen
FillShape
Flush
Forwd
Frac
FreeMem
GetDir
GetDotColor
GetMem
GetPic
GotoXY
GraphBackground
GraphColorMode
Graphics
GraphMode
GraphWindow
Green
Halt
Heading
HeapPtr
Hi
HideTurtle
HiRes
HiResColor
Home
Input
InsLine
Insert
Int
Integer
IOresult
Kbd
KeyPressed
Length
LightBlue
LightCyan
LightGray
LightGreen
LightRed
LightMagenta
Ln
Lo
LongFilePosition
LongFileSize
LongSeek
LowVideo
Lst
LstOutPtr
Magenta
Mark
MaxAvail
MaxInt
Mem
MemAvail
MkDir
Move
MsDos
New
NormVideo
NoSound
NoWrap
Odd
Ord
OutPut
OvrDrive
OvrPath
Palette
ParamCount
ParamStr
Pattern
PenDown
PenUp
Pi
Plot
Port
Pos
Pred
Ptr
PutPic
Random
Randomize
Read
Readln
Real
Red
Release
Rename
Reset
Rewrite
RmDir
Round
Seek
SetHeading

SetPenColor	*Str*	*TurnLeft*	*WhereX*
SetPosition	*Succ*	*TurnRight*	*WhereY*
ShowTurtle	*Swap*	*TurtleDelay*	*White*
Sin	*Text*	*TurtleThere*	*Window*
SizeOf	*TextBackground*	*TurtleWindow*	*Wrap*
SeekEof	*TextColor*	*UpCase*	*Write*
SeekEoln	*TextMode*	*Usr*	*Writeln*
Sound	*Trm*	*UsrInPtr*	*XCor*
Sqr	*True*	*UsrOutPtr*	*YCor*
Sqrt	*Trunc*	*Val*	*Yellow*
	Truncate		

Jeśli w programie zostanie zadeklarowany identyfikator występujący na przytoczonej liście. to w zasięgu tej deklaracji zostanie przesłonięte pierwotne znaczenie danego identyfikatora.

Przykład
Przesłonięcie znaczenia identyfikatora

```
program jb;
const
  false = true;
begin
  Writeln(false)
end.
```

- Wykonanie programu powoduje wyprowadzenie napisu *TRUE*.
- Gdyby z programu usunięto tekst między pierwszym średnikiem a słowem **begin** (wyłącznie), to wykonanie programu spowodowałoby wyprowadzenie napisu *FALSE*. □

Literały

Literałem jest napis reprezentujący stałą — obiekt, który podczas wykonywania programu nie ulega zmianie.
Literały dzielą się na *arytmetyczne*, *łańcuchowe* i *logiczne*. Literały arytmetyczne dzielą się na całkowite i rzeczywiste. Literały te będą nazywane *liczbami*.

Literały całkowite

Podstawowy *literał całkowity* składa się z ciągu cyfr dziesiętnych, który może być poprzedzony znakiem + (plus) lub — (minus). Wartość liczbowa literału całkowitego musi się mieścić w przedziale domkniętym [—32768 ; 32767].

W języku Turbo Pascal zapewniono możliwość przedstawiania literałów całkowitych i bajtowych jako liczb szesnastkowych. W takim przypadku literał składa się ze znaku $ (dolar), po którym następują cyfry szesnastkowe. Przed znakiem $ może występować znak + (plus) albo − (minus).

Przykład
Kilka poprawnych i niepoprawnych literałów całkowitych

Literały całkowite: 23
−23
002
$*ABC*
−$2*B*

Napisy, które nie są literałami całkowitymi:

2*B*6 – znak *B* nie jest cyfrą dziesiętną
$3*G* – znak *G* nie jest cyfrą szesnastkową
2.3 – znak . nie jest cyfrą □

Literały rzeczywiste

Literał rzeczywisty składa się z następujących po sobie: części całkowitej, kropki, części ułamkowej, małej albo dużej litery *E* oraz wykładnika. Część całkowita, ułamkowa i wykładnik składają się z ciągów cyfr dziesiętnych, a część całkowitą i cyfry wykładnika może poprzedzać znak + (plus) albo − (minus). Część ułamkową wraz z kropką, albo wykładnik wraz z literą *E*, można pominąć.

Przykład
Kilka poprawnych i niepoprawnych literałów rzeczywistych

Literały rzeczywiste: −2.3
2.5*e*2
4.0*E*−3
2*E*3

Napisy, które nie są literałami rzeczywistymi:

3*e*2.0 – kropka w wykładniku
.25 – brak części całkowitej
2.3*D*2 – błędny wykładnik □

Literały łańcuchowe

Literały łańcuchowe reprezentują ciągi znaków. Jeśli literał łańcuchowy reprezentuje jeden znak, to jest nazywany *literałem znakowym*. W ogólnym przypadku literał łańcuchowy składa się z sekwencji znaków zawartych między

parą apostrofów, znaków sterujących i znaków wyrażonych dziesiętnie i szesnastkowo. W sekwencji zawartej między parą apostrofów mogą występować dowolne znaki widoczne, wraz ze spacją, a znak ' (apostrof) jest reprezentowany przez parę sąsiadujących apostrofów. Znaki sterujące są przedstawiane za pomocą pary znaków, z których pierwszy jest znakiem ^ (caret), a drugi jest znakiem widocznym. Reprezentacja tak przedstawionego znaku sterującego jest identyczna z reprezentacją znaku uzyskanego przez jednoczesne naciśnięcie klawisza Ctrl i danego znaku widocznego. Znaki wyrażone dziesiętnie składają się ze znaku # (hash), po którym następuje dziesiętny kod znaku, a znaki wyrażone szesnastkowo składają się z następujących bezpośrednio po sobie: znaku # (hash), znaku $ (dolar) i szesnastkowego kodu znaku.

Przykład
Kilka poprawnych i niepoprawnych literałów łańcuchowych

Literały łańcuchowe: '*jb*' (dwa znaki)
'''' (jeden znak)
^*G* (jeden znak)
#65 (jeden znak)
#$41 (jeden znak)
^*G*'*jb*' (trzy znaki)
' ' (dwa znaki spacji)
^*G*^*G*^*G*'*hello*'#13#10'*jan*' (13 znaków)

Napisy, które nie są literałami łańcuchowymi:

''' — brak apostrofu
#256 — kod większy niż 255
#$4*G* — *G* nie jest cyfrą szesnastkową □

Literały logiczne

Literały logiczne mają postać napisów *true* i *false*. Pierwszy z nich reprezentuje daną logiczną „prawda", a drugi — daną logiczną „fałsz".

Przykład
Kilka poprawnych i niepoprawnych literałów logicznych

Literały logiczne: *false*
False
True

Napisy, które nie są literałami logicznymi:

prawda — błędny napis
fałsz — błędny napis □

Komentarze

Komentarzem jest napis rozpoczynający się od nawiasu klamrowego otwierającego, zakończony najbliższym nawiasem klamrowym zamykającym. Rolę nawiasów klamrowych: { (otwierającego) i } (zamykającego) mogą pełnić dwuznaki (* i *). Komentarze ograniczone znakami { i } mogą być zagnieżdżone w komentarzach ograniczonych dwuznakami (* i *), a komentarze ograniczone dwuznakami (* i *) mogą być zagnieżdżone w komentarzach ograniczonych znakami { i }.

Jeśli bezpośrednio po znaku albo dwuznaku rozpoczynającym komentarz występuje znak $ (dolar), to komentarz taki zostaje uznany za *dyrektywę kompilatora*. Dyrektywa taka nie może znajdować się w komentarzu zagnieżdżonym.

Przykład
Kilka poprawnych i niepoprawnych komentarzy i dyrektyw kompilatora

Komentarze: (* *data* *)
{ *data* }
(* *out* {*file*} *)
Dyrektywy: (*$*i include.set* *)
{$*v*,*b*+}

Napisy, które nie są komentarzami:

(* *inside*} – niepoprawne ograniczenie komentarza
{{*inside*}} – niepoprawne zagnieżdżenie

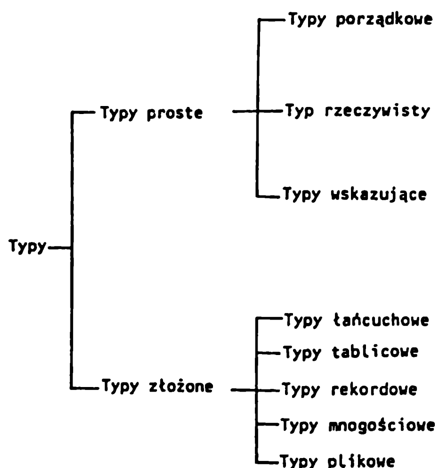
Napisy, które nie są dyrektywami kompilatora:

{$*v*–} – spacja między { i $
{$ *v*–}– spacja między $ i *v*– □

3. Typy standardowe

Pojęcie typu jest związane z ustalonym zbiorem danych. W tym sensie zmienna jest pewnego typu, jeśli można jej przypisywać dane należące do zbioru związanego z tym typem. W języku Turbo Pascal wymaga się jawnego określenia typu każdej ze zmiennych programu.

Związanie nazwy zmiennej z wybranym typem odbywa się w *deklaracji zmiennej*. *Typy zmiennych*, sklasyfikowane jak na rys. 3.1, dzielą się na *proste* i *złożone*. Typami prostymi są typy *porządkowe*, *rzeczywiste*, *łańcuchowe* i



Rys. 3.1 Klasyfikacja typów

wskazujące, a typami złożonymi — typy *tablicowe*, *rekordowe*, *mnogościowe* i *plikowe*. Wśród wymienionych typów można wyróżnić typy standardowe: *integer*, *byte*, *char*, *boolean* i *real*. Pierwsze cztery będą również nazywane *typami porządkowymi*. Mają one tę właściwość, że każdy z nich jest związany ze zbiorem przeliczalnym.

Typy standardowe są predefiniowane, tzn. nie wymagają definiowania. Zasięgiem ich niejawnych definicji jest cały program, z wykluczeniem tych jego fragmentów, w których wprowadzono jawne deklaracje przesłaniające.

Typ całkowity (*integer*)

Dana typu *integer* jest elementem podzbioru liczb całkowitych. Wartości tych liczb należą do przedziału domkniętego [−32768 ; 32767]. Każda z danych typu *integer* jest reprezentowana w 2 bajtach pamięci.

Typ bajtowy (*byte*)

Dana typu byte jest elementem podzbioru liczb całkowitych. Wartości danych typu byte należą do przedziału domkniętego [0 ; 255]. W prawie każdym miejscu programu, w którym występuje odwołanie do danej bajtowej może wystąpić odwołanie do danej typu *integer* i odwrotnie. Ważny wyjątek od tej zasady dotyczy skojarzeń przez wskazanie parametrów i argumentów procedur i funkcji. W tym przypadku wymaga się pełnej zgodności typów danych. Każda z danych typu byte jest reprezentowana w 1 bajcie pamięci.

Typ znakowy (*char*)

Dana typu *char* jest elementem zbioru znaków kodu ASCII. Każda z danych typu *char* jest reprezentowana w 1 bajcie pamięci.

Typ logiczny (*boolean*)

Dana typu *boolean* jest elementem dwuelementowego zbioru danych logicznych. Do oznaczenia tych danych służą standardowe identyfikatory *true* i *false*. Każda z danych typu *boolean* jest reprezentowana w 1 bajcie pamięci.

Typ rzeczywisty (*real*)

Dana typu *real* jest elementem zbioru liczb rzeczywistych. Do zbioru tego należy liczba 0 oraz liczby dodatnie i ujemne, których wartości bezwzględne należą do przedziału $[10^{-38}; 10^{38}]$. Każda z danych typu *real* jest reprezentowana w 6 bajtach pamięci, z dokładnością do ok. 11 cyfr znaczących.

4. Programy

Program napisany w języku Turbo Pascal składa się z *nagłówka programu*, *bloku* i *znaku* . (*kropka*). W nagłówku jest określona *nazwa programu* oraz ujęty w nawiasy okrągłe *wykaz identyfikatorów*, opisujący sposób komunikowania się programu z otoczeniem. Blok składa się z *części deklaracyjnej* i *wykonawczej*. Zarówno nagłówek, jak i część deklaracyjna bloku mogą być pominięte.

Nagłówek programu

Jeśli program zaczyna się od nagłówka, to występuje w nim przynajmniej nazwa programu. Po nazwie tej może wystąpić — ujęta w nawiasy okrągłe — lista parametrów programu. Lista ta wyszczególnia identyfikatory plików, za pośrednictwem których program komunikuje się z otoczeniem. Liście tej nie jest nadawana żadna interpretacja i z tego względu może być (wraz z otaczającymi ją nawiasami) pominięta.

Składnia

nagłówek-programu:
program *nazwa-programu* (*lista-nazw-plików*) ;
nazwa-programu:
identyfikator
nazwa-pliku:
identyfikator

Przykłady

program *Volume*;
program *Lister*(*Output*);
program *Copier*(*Input*,*Output*); □

Blok

Blok składa się z *części deklaracyjnej* i *części wykonawczej*. W części deklaracyjnej znajdują się definicje typów, definicje nazw literałów, definicje i deklaracje podprogramów, a także deklaracje etykiet i zmiennych. Wszystkie te deklaracje i definicje mogą być podane w dowolnym porządku, a poszczególne grupy deklaracji i definicji mogą się przeplatać. Należy rozumieć to w taki sposób, że np. po definicjach typów mogą następować deklaracje zmiennych, a po nich ponownie definicje typów i znowu deklaracje zmiennych. Część wykonawcza skłąda się z instrukcji złożonej zawierającej instrukcje programu.

Składnia

blok:
  część-deklaracyjna część-wykonawcza
część-deklaracyjna:
  wykaz-deklaracji-i-definicji
część-wykonawcza:
  instrukcja-złożona
deklaracje-i-definicje:
  deklaracje-etykiet
  definicje-nazw-literałów
  definicje-typów
  deklaracje-zmiennych
  definicje-i-deklaracje-podprogramów
  inicjacje

Deklaracje etykiet

Każda instrukcja czynna programu może być opatrzona *etykietą* umożliwiającą odwołanie się do niej w instrukcji goto. Etykieta składa się z identyfikatora albo z ciągu cyfr. Bezpośrednio po etykiecie następuje znak : (dwukropek) oddzielający ją od innej etykiety albo instrukcji. Zanim etykieta zostanie użyta w programie, jej nazwa musi być zadeklarowana. Deklaracje-etykiet składają się ze słowa kluczowego **label**, po którym następuje lista napisów identycznych z nazwami etykiet.

Składnia

deklaracje-etykiet:
  label *lista-etykiet* ;

Przykłady

  label 10, *abort*, *quit*;
  label 99999; 

Definicje nazw literałów

W celu zwiększenia czytelności programu pożądane jest niekiedy zastąpienie literałów odpowiednio dobranymi identyfikatorami reprezentującymi te literały. W takich przypadkach każde wystąpienie identyfikatora jest równoważne wystąpieniu związanego z nim literału. W języku Turbo Pascal związanie identyfikatorów z literałami odbywa się podczas interpretowania *definicji nazw literałów*. Każda z definicji nazw literałów składa się ze słowa kluczowego **const**, po którym następuje sekwencja przypisań. Każde z przypisań składa się z identyfikatora, bezpośrednio po którym następuje znak = (równa się), wyrażenie stałe oraz znak ; (średnik). *Wyrażeniem stałym* może być liczba, literał łańcuchowy, nazwa literału albo nazwa literału poprzedzona znakiem – (minus). Predefiniowanymi nazwami literałów są

Pi – typu *real*, reprezentujące literał 3.1415926536
MaxInt – typu *integer*, reprezentujące literał 32767

Składnia

definicje-nazw-literałów:
const *sekwencja-przypisań*
przypisanie:
identyfikator = *liczba* ;
identyfikator = *literał-łańcuchowy* ;
identyfikator = *literał-logiczny* ;
identyfikator = *nazwa-literału* ;
identyfikator = *znak nazwa-literału* ;

Przykłady

const *e* = 2.718282;
g = 9.81;
const *TwoChar* = '2';
Two = 2;
Minus2 = – *Two*;
const *FirstName* = '*janek*'; □

Definicje typów

Podczas deklarowania zmiennych programu następuje określenie ich *typu*. Typ zmiennej może należeć do zbioru predefiniowanych typów standardowych, takich jak np. *boolean* lub *real*, ale może także zostać zdefiniowany odrębnie w *definicji typu*. W ogólnym przypadku definicje-typów składają się ze słowa kluczowego **type**, po którym następuje sekwencja definicji typów. Każda z definicji typu składa się z identyfikatora typu definiowanego, znaku = (równa się), opisu typu i znaku ; (średnik).

Składnia

definicje-typów:
  type *sekwencja-definicji-typu*

definicja-typu:
  typ-definiowany = *opis-typu*;

typ-definiowany:
  identyfikator

opis-typu:
  opis-typu-standardowego
  opis-typu-definiowanego

opis-typu-standardowego:
  identyfikator-typu-standardowego

opis-typu-definiowanego:
  opis-typu-wyliczeniowego
  opis-typu-okrojonego
  opis-typu-łańcuchowego
  opis-typu-tablicowego
  opis-typu-rekordowego
  opis-typu-mnogościowego
  opis-typu-plikowego
  opis-typu-wskazującego

Przykłady

```
type
  Number = integer;
  Color = (Red,Green,Blue,Orange);
  RGB = (Red..Blue);
  Name = string[20];
  Matrix = array[1..5,1..5] of real;
  Person = record
             FirstName : string[6];
             LastName : Name;
             Salary : real
           end;
  Digits = set of 0..9;
  DataFile = file of Person;
  Ref = ^Matrix;
```
□

Deklaracje zmiennych

Każda zmienna programu, zarówno prosta jak i agregatowa, musi być zadeklarowana przed jej użyciem. *Deklaracje zmiennych* składają się ze słowa

kluczowego **var**, po którym następuje wykaz deklaracji. *Elementem wykazu deklaracji* jest sekwencja napisów składająca się z listy identyfikatorów, znaku : (dwukropek), opisu typu i znaku ; (średnik). Zinterpretowanie każdej z deklaracji powoduje utworzenie zmiennych podanego typu.

Składnia

deklaracje zmiennych:
var *wykaz-deklaracji*
deklaracja:
lista-identyfikatorów : *opis-typu*;

Przykłady

```
var
  Length, Area, Volume : real;
  Count : integer;
  Buffer : array[0..255] of byte;
  Rejected : boolean;
```
□

Zakresem deklaracji zmiennej jest blok, w którym zmienna ta została zadeklarowana. Jeśli blok ten zawiera inne bloki, w których występują deklaracje zmiennych oznaczone takim samym identyfikatorem, to zasięg deklaracji jest mniejszy niż zakres i nie obejmuje bloków wewnętrznych, w których wystąpiły deklaracje przesłaniające. Zakres deklaracji zmiennej rozpoczyna się nie od początku bloku, lecz od miejsca zadeklarowania jej identyfikatora.

Przykład
Zakres i zasięg deklaracji

	zakres		zasięg	
program *Tricky*;	1		1	
var	1		1	
Toggle : ($false^1$..*true*);	1		1	
$false^2$: *integer*;	1	2		2
begin	1	2		2
false := 5;	1	2		2
Writeln(*false*)	1	2		2
end.				

● Zasięgiem predefiniowanego identyfikatora *false* jest jego zakres pomniejszony o zakres jawnie zadeklarowanego identyfikatora *false* typu *integer*.
● Zakres i zasięg zadeklarowanego jawnie identyfikatora *false* jest taki sam.
● Ponieważ wywołanie procedury *Writeln* występuje w zasięgu jawnie zadeklarowanego identyfikatora *false*, wykonanie programu powoduje wyprowadzenie liczby 5. □

Deklaracje i definicje podprogramów

Podprogramy dzielą się na *funkcje* i *procedury*. *Definicja podprogramu* określa czynności, jakie zostaną wykonane bezpośrednio po wywołaniu podprogramu. W przypadku funkcji określa również typ rezultatu funkcji udostępnionego w miejscu jej wywołania.

Użycie *deklaracji podprogramu* jest niezbędne jedynie wtedy, kiedy nie jest możliwe takie uporządkowanie definicji podprogramów, aby odwołania do tych podprogramów występowały w zasięgu ich definicji.

Przykład Wzajemne wywołania procedur

```
program jb;
  ...
  procedure second; forward;
  procedure first;
  begin
    ...
    second
  end;
  procedure second;
  begin
    ...
    first
  end;
begin
  ...
end.
```

• Ponieważ w definicji procedury *first*

```
procedure first;
begin
  ...
  second
end;
```

występuje wywołanie procedury *second*, a w definicji procedury *second* występuje wywołanie procedury *first*, konieczne było użycie deklaracji procedury *second* o postaci

```
procedure second; forward;
```

• W odróżnieniu od definicji deklaracja służy jedynie zaprezentowaniu identyfikatora podprogramu, a jeśli podprogram ma parametry, to także jego parametrów. □

5. Wyrażenia

Wyrażenia są napisami określającymi czynności, jakie mają być wykonane w celu utworzenia danej. Wykonanie tych czynności będzie nazywane *opracowaniem wyrażenia.* Ponieważ rezultatem opracowania wyrażenia jest dana, więc samo wyrażenie można uważać za napis reprezentujący daną. W niniejszym rozdziale zostaną opisane zasady opracowywania wyrażeń, których elementy składowe reprezentują dane typu *integer*, *real*, *boolean* i *char.* Opisy wyrażeń zawierających odwołania do danych typów definiowanych zostaną przedstawione podczas omawiania tych typów.

Operatory

Operatory dzielą się na *jednoargumentowe* i *dwuargumentowe.* Innym kryterium ich podziału jest *priorytet.* W kolejności malejącego priorytetu ogół operatorów można podzielić na

a) Jednoargumentowy operator zmiany znaku

    −

b) Jednoargumentowy operator negacji

    not

c) Dwuargumentowe operatory czynnikowe

    * / **div mod and shl shr**

d) Dwuargumentowe operatory składnikowe

    + − **or xor**

e) Dwuargumentowe operatory relacyjne

    = <> < > <= >= **in**

Jeśli pewnego podwyrażenia dotyczą dwa operatory o równym priorytecie, to stosowne operacje są wykonywane od lewej do prawej. Podwyrażenia ujęte w nawiasy okrągłe są opracowywane w pierwszej kolejności. Z tego względu nawiasy okrągłe mogą być traktowane jako operator jednoargumentowy o najwyższym priorytecie.

Jeśli argumentami operacji czynnikowych i składnikowych są podwyrażenia reprezentujące dane typu *integer* lub *real*, to rezultat operacji jest typu *integer* tylko wtedy, kiedy oba argumenty są typu *integer*, a operacją nie jest dzielenie. W pozostałych przypadkach rezultat operacji jest typu *real*.

Przykład

Wyznaczanie danej reprezentowanej przez wyrażenie

2+3/4

• Ponieważ priorytet dzielenia jest wyższy niż priorytet dodawania, przytoczone wyrażenie jest traktowane tak jak wyrażenie

2+(3/4)

a nie jak wyrażenie

(2+3)/4

• Rezultatem opracowania wyrażenia 3/4 jest dana typu *real* o wartości równej — w przybliżeniu — wartości literału 0.75.

• Przytoczone wyrażenie reprezentuje daną typu *real* o wartości bliskiej wartości danej reprezentowanej przez literał 2.75. □

Operator zmiany znaku

Wykonanie jednoargumentowej operacji — (minus) powoduje zmianę znaku danej na przeciwny. Jeśli dana ma wartość 0, to rezultatem operacji jest także dana o wartości 0. Argumentami operacji — (minus) mogą być tylko dane typu *real* i *integer*.

Przykłady

Wyrażenie	Rezultat
−(−4)	4
−3/2	−1.5

Operator negacji

Wykonanie jednoargumentowej operacji **not** może dotyczyć jedynie danych typu *boolean* i *integer*. W pierwszym przypadku dana o wartości *false* zostanie

przekształcona w daną o wartości true i odwrotnie. W drugim, każdy bit reprezentacji danej zostanie zmieniony na przeciwny.

Przykłady

Wyrażenie	Rezultat
not *false*	*true*
not *true*	*false*
not 0	−1
not $FFFF	0
not −2	1
not −32768	32767

□

Operatory czynnikowe

Dwuargumentowe operatory * (mnożenia) i / (dzielenia) mogą dotyczyć zarówno danych typu *real*, jak i danych typu *integer*. Dwuargumentowy operator **and** może dotyczyć zarówno danych typu *boolean*, jak i danych typu *integer*. Pozostałe operatory czynnikowe mogą dotyczyć jedynie danych typu *integer*. Wymagania te zebrano w tabl. 5.1, w której przytoczono także typ rezultatu operacji.

Tablica 5.1. Operatory czynnikowe

Operator	Argument – 1	Argument – 2	Rezultat
*	*real*	*real*	*real*
	real	*integer*	*real*
	integer	*real*	*real*
	integer	*integer*	*integer*
/	*real*	*real*	*real*
	real	*integer*	*real*
	integer	*real*	*real*
	integer	*integer*	*real*
div	*integer*	*integer*	*integer*
mod	*integer*	*integer*	*integer*
and	*integer*	*integer*	*integer*
	boolean	*boolean*	*boolean*
shl	*integer*	*integer*	*integer*
shr	*integer*	*integer*	*integer*

Sposób wykonywania operacji * i / nie wymaga wyjaśnień. Należy jedynie pamiętać, że rezultatem dzielenia jest zawsze dana typu *real*.

Przykłady

Wyrażenie	Rezultat
25*40	1000
25*40.0	1000.0
0.25*40.0	10.0
24/3	8.0
24.0/3.0	8.0

□

Rezultatem operacji **div** jest dana typu *integer*, o tak dobranej wartości, że stanowi ona część całkowitą rezultatu podzielenia pierwszego argumentu przez drugi. Operacja **mod** jest natomiast zdefiniowana jako

a **mod** *b* = *a*−((*a* **div** *b*)*b)

Przykłady

Wyrażenie	Rezultat
25 **div** 7	3
−25 **div** 7	−3
25 **mod** 7	4
−25 **mod** 7	−4

□

Rezultatem operacji **and** na danych typu *boolean* jest koniunkcja tych argumentów wyznaczona jak następuje:

argument *a*	argument *b*	*a* **and** *b*
false	*false*	*false*
true	*false*	*false*
false	*true*	*false*
true	*true*	*true*

Rezultatem operacji **and** na danych typu *integer* jest dana typu *integer*, której reprezentacja stanowi iloczyn logiczny wyznaczony równolegle na wszystkich odpowiadających sobie parach bitów argumentów. Iloczyn logiczny dla pary bitów ma wartość 1 tylko wtedy, kiedy oba bity mają wartość 1. W przeciwnym razie ma on wartość 0.

Przykłady

Wyrażenie	Rezultat
false **and** *true*	*false*
2 **and** 2	2
12 **and** 22	4
−5 **and** 0	0

□

Dwuargumentowe operacje **shl** i **shr** mogą dotyczyć jedynie danych typu *integer*. Rezultatem każdej z tych operacji jest również dana typu *integer*. Dana ta ma taką reprezentację, jaka powstaje z reprezentacji pierwszego argumentu

po przesunięciu go w lewo (**shl**) albo w prawo (**shr**), o taką liczbę pozycji, jaką określa wartość drugiego argumentu. Przyjmuje się, że przesuwanie ma charakter logiczny. Oznacza to, że podczas przesuwania bit znaku nie ulega powieleniu, a zwolnione pozycje są zapełniane bitami 0.

Przykłady

Wyrażenie	Rezultat
2 **shl** 1	4
2 **shl** 3	16
3 **shr** 1	1
−3 **shr** 1	32766

□

Operatory składnikowe

Dwuargumentowe operatory + (dodawania) i − (odejmowania) mogą dotyczyć zarówno danych typu *real*, jak i danych typu *integer*. Pozostałe operatory typu dodawania mogą dotyczyć par argumentów identycznych typów: *integer* albo *boolean*. Wymagania te zebrano w tabl. 5.2, w której przytoczono także typ rezultatu operacji.

Tablica 5.2. Operatory składnikowe

Operator	Argument−1	Argument−2	Rezultat
+	*real*	*real*	*real*
	real	*integer*	*real*
	integer	*real*	*real*
	integer	*integer*	*integer*
−	*real*	*real*	*real*
	real	*integer*	*real*
	integer	*real*	*real*
	integer	*integer*	*integer*
or	*integer*	*integer*	*integer*
	boolean	*boolean*	*boolean*
xor	*integer*	*integer*	*integer*
	boolean	*boolean*	*boolean*

Sposób wykonywania operacji + i − nie wymaga wyjaśnień. Należy jedynie pamiętać, że rezultat operacji jest typu *integer* tylko wtedy, kiedy oba argumenty są tego typu. W przeciwnym razie rezultat operacji jest typu *real*.

Przykłady

Wyrażenie	Rezultat
20+40	60
20.0+40	60.0
60.0−40.0	20.0

□

Rezultatem operacji **or** na danych typu *boolean* jest alternatywa tych argumentów, wyznaczona jak następuje:

argument *a*	argument *b*	*a* **or** *b*
false	*false*	*false*
true	*false*	*true*
false	*true*	*true*
true	*true*	*true*

Rezultatem operacji **or** na danych typu *integer* jest dana typu *integer*, której reprezentacja stanowi sumę logiczną wyznaczoną równolegle na wszystkich odpowiadających sobie parach bitów argumentów. Suma logiczna dla pary bitów ma wartość 0 tylko wtedy, kiedy oba bity mają wartość 0. W przeciwnym razie ma wartość 1.

Przykłady

Wyrażenie	Rezultat
false **or** *true*	*true*
2 **or** 3	3
12 **or** 22	30
$8000 **or** 1	−32767

□

Rezultatem operacji **xor** na danych typu *boolean* jest różnica symetryczna tych argumentów, wyznaczona jak następuje:

argument *a*	argument *b*	*a* **xor** *b*
false	*false*	*false*
true	*false*	*true*
false	*true*	*true*
true	*true*	*false*

Rezultatem operacji **xor** na danych typu *integer* jest dana typu *integer*, której reprezentacja stanowi sumę modulo 2 wyznaczoną równolegle na wszystkich odpowiadających sobie parach bitów argumentów. Suma modulo 2 ma wartość 0 tylko wtedy, kiedy oba bity są identyczne. W przeciwnym razie ma wartość 1.

Przykłady

Wyrażenie	Rezultat
true **xor** *true*	*false*
$8000 **xor** 2	−32766
12 **xor** 22	26
−1 **xor** −1	0

□

Relacje

Operatory *relacji* mogą dotyczyć dowolnych danych typu *integer*, *boolean*, *real* i *char*, a także danych łańcuchowych, wskazujących i mnogościowych. Rezultatem operacji jest zawsze dana typu boolean, o wartości *false* albo *true*. Jeśli pewien argument relacji jest typu *boolean* albo *char*, to drugi musi być takiego samego typu. W przypadku argumentów arytmetycznych nie wymaga się identyczności typów. Umożliwia to np. porównywanie argumentów typu *real* z argumentami typu *integer*.

Operatorami relacji są: = (równe), <> (nie równe), > (większe), < (mniejsze), > = (większe lub równe), < = (mniejsze lub równe).

Przykłady

Wyrażenie	Rezultat
20 = 20	*true*
30.0 > 20	*true*
'*J*' < '*B*'	*false*
false <> *true*	*true*

□

Wywołania funkcji

Wywołanie funkcji składa się z identyfikatora funkcji, bezpośrednio po którym występuje ujęta w nawiasy okrągłe lista argumentów wywołania. Wywołanie funkcji bezparametrowej składa się tylko z identyfikatora funkcji.

Przykłady

Eof (*Input*)
KeyPressed □

6. Instrukcje

Instrukcja programu stanowi opis czynności, które będą wykonane podczas realizowania algorytmu. W Turbo Pascalu istnieją dwa rodzaje instrukcji: *proste* i *strukturalne*. Instrukcjami prostymi są: instrukcja przypisania, instrukcja wywołania procedury, instrukcja przejścia i instrukcja pusta. Instrukcjami strukturalnymi są: instrukcja grupująca, instrukcja warunkowa, instrukcja wyboru, trzy instrukcje iteracyjne oraz instrukcja wiążąca. Każda para instrukcji programu jest oddzielona znakiem ; (średnik).

Instrukcja przypisania

Instrukcja przypisania składa się z następujących po sobie: napisu identyfikującego zmienną, dwuznaku przypisania i wyrażenia. W obrębie definicji funkcji musi wystąpić co najmniej jedna instrukcja przypisania, w której napisem identyfikującym zmienną jest identyfikator funkcji.

Wykonanie instrukcji przypisania powoduje wyznaczenie danej reprezentowanej przez wyrażenie i przypisanie tej danej zmiennej identyfikowanej przez napis występujący po lewej stronie dwuznaku przypisania. W przypadku omówionego wyżej przypisania występującego w obrębie definicji funkcji następuje określenie *rezultatu funkcji*, tj. danej udostępnianej w miejscu wywołania funkcji.

Wymaga się, aby typ wyrażenia oraz typ zmiennej były zgodne w sensie przypisania. Zgodność ta jest zapewniona wtedy, kiedy zmienna i wyrażenie są tego samego typu, jak również wówczas, gdy zmienna jest typu *real*, a wyrażenie jest typu *integer*. Należy nadmienić, że nie jest dozwolone przypisywanie plików, ale jest dozwolone przypisywanie takich danych strukturalnych jak tablice, rekordy i mnogości.

Składnia
instrukcja-przypisania:
zmienna := *wyrażenie*

Przykłady

i := *i*+2
x1 := (−*b*+*sqrt*(*b*∗*b*−4∗*a*∗*c*))/(2∗*a*)
arr[2,3] := *m* = *n*
fun := *false* □

Instrukcja procedury

Instrukcja procedury składa się z identyfikatora procedury, bezpośrednio po którym następuje ujęta w nawiasy okrągłe lista argumentów wywołania. Instrukcja procedury bezparametrowej składa się tylko z identyfikatora procedury.

Wykonanie instrukcji procedury powoduje wykonanie czynności wyszczególnionych w definicji tej procedury. Zanim to nastąpi, odbywa się identyfikacja zmiennych stanowiących argumenty wywołania procedury i wyznaczenie wartości wyrażeń występujących w wywołaniu.

Składnia
instrukcja-procedury:
nazwa-procedury (*lista-argumentów*)
nazwa-procedury
nazwa-procedury:
identyfikator

Przykłady

Rewrite(*OutFile*)
Move(*source*,*target*)
Exit □

Instrukcja przejścia

Instrukcja przejścia składa się ze słowa kluczowego **goto**, bezpośrednio po którym następuje nazwa etykiety opatrującej instrukcję.

Wykonanie instrukcji przejścia powoduje przejście do wykonywania ciągu instrukcji rozpoczynającego się od instrukcji opatrzonej podaną etykietą. Wymaga się, aby instrukcja przejścia powołująca się na etykietę znajdowała się

w zasięgu deklaracji tej etykiety. Zabrania się, aby wykonanie instrukcji przejścia prowadziło do wnętrza instrukcji strukturalnej albo podprogramu. Zabrania się także, aby wykonanie takiej instrukcji powodowało przedwczesne zakończenie wykonywania podprogramu.

Składnia
    instrukcja-przejścia:
        goto *etykieta*
    etykieta:
        identyfikator
        ciąg-cyfr

Przykłady
        goto 345
        goto *finish* □

Instrukcja pusta

Instrukcja pusta jest jedyną instrukcją, której zapisanie nie wymaga użycia jednostek leksykalnych. Występuje ona w tych kontekstach, w których jest wymagane użycie instrukcji, ale chce się uniknąć wykonania jakiejkolwiek akcji.

Wykonanie instrukcji pustej nie wywołuje żadnych skutków.

Składnia
    instrukcja-pusta:
        puste
    puste:

Przykłady
(Miejsce wystąpienia instrukcji pustej oznaczono komentarzem zawierającym wykrzyknik)

        begin { ! } **end**
        while *false* **do** { ! };
        repeat { ! } **until** **true**
        begin { ! } ; { ! } **end**
        case *true* **of** *false* {!}; *true:* { ! } **end** □

Instrukcja grupująca

Instrukcja grupująca składa się ze słowa kluczowego **begin**, bezpośrednio po którym następuje sekwencja instrukcji i słowo kluczowe **end**. Instrukcja

grupująca jest używana wszędzie tam, gdzie składnia języka wymaga użycia jednej instrukcji, a niezbędne jest wykonanie ich sekwencji.

Wykonanie instrukcji grupującej powoduje wykonanie zawartej w niej sekwencji instrukcji.

Składnia

instrukcja-grupująca:

begin *sekwencja-instrukcji* **end**

Przykłady

```
begin
  i := 2;    j := 2
end

begin
  Writeln(OutFile);
  Close(OutFile)
end
```

□

Instrukcja warunkowa

Instrukcja warunkowa składa się ze słowa kluczowego **if**, bezpośrednio po którym następują kolejno: wyrażenie logiczne, słowo kluczowe **then** i instrukcja. Po instrukcji może występować słowo kluczowe **else** i ponownie instrukcja.

Wykonanie instrukcji warunkowej składa się z wyznaczenia wartości wyrażenia, a następnie, jeśli rezultat jest daną o wartości *true*, wykonania instrukcji następującej po słowie kluczowym **then**. Jeśli rezultat jest daną o wartości *false*, a instrukcja warunkowa nie zawiera słowa kluczowego **else**, to jej wykonanie uznaje się za zakończone. Jeśli zawiera takie słowo, to jest wykonywana instrukcja występująca po tym słowie.

Jeśli instrukcja występująca po słowie kluczowym **if** jest instrukcją warunkową, to musi mieć postać ze słowem kluczowym **else**.

Składnia

instrukcja-warunkowa:

if *wyrażenie-logiczne* **then** *instrukcja*

if *wyrażenie-logiczne* **then** *instrukcja*
else *instrukcja*

Przykłady

```
if a = b then a := 5 else b := 5
```

```
if a = b then begin
a := 6;
b := 6
end

if a = b then
  if a = 5 then
    b := 2
  else
    a := 3
else
  a := 13
```

Instrukcja wyboru

Instrukcja wyboru składa się ze słowa kluczowego **case**, bezpośrednio po którym występuje wyrażenie selekcyjne, słowo kluczowe **of**, sekwencja instrukcji wyboru poprzedzonych etykietami wyboru, a po niej słowo kluczowe **end**.

Lista etykiet wyboru składa się z literałów, a listę oddziela od instrukcji znak : (dwukropek). Każdy z literałów listy musi być tego samego typu jak wyrażenie selekcyjne. Jeśli pewna podlista listy literałów stanowi ciąg kolejnych elementów typu porządkowego, to może być zastąpiona konstrukcją

pierwszy..ostatni

wyszczególniającą *pierwszy* i *ostatni* element tego ciągu. Zamiast etykiet wyboru poprzedzających ostatnią funkcję sekwencji instrukcji wyboru może być użyte słowo kluczowe **else**, a po nim dowolny wykaz instrukcji. Po takiej „etykiecie" nie stawia się dwukropka.

Składnia

instrukcja-wyboru:
case *wyrażenie-selekcyjne* **of**
wykaz-instrukcji-wyboru **end**

Wykonanie instrukcji wyboru składa się z wyznaczenia wartości wyrażenia selekcyjnego, a następnie wykonania tej instrukcji należącej do wykazu instrukcji wyboru, która jest poprzedzona etykietą wyboru o wartości równej wartości wyrażenia selekcyjnego. Jeśli takiej instrukcji nie ma, to są wykonywane instrukcje poprzedzone etykietą wyboru ze słowem kluczowym **else**, a jeśli i takiej nie ma, to jest wykonywana instrukcja pusta.

Przykłady

```
case a = b of
  false: i := 5;
  true: j := 2
end

case Operator of
  '+': Result := Result + 1;
  '-': Result := Result - 1
  else goto fin
end

case Year of
  1945..1955 : Writeln('hard work');
  1956..1969 : Writeln('hopes');
  1970..1979 : Writeln('prosperity');
  1980..1981 : Writeln('euphoria')
  else         Writeln('no comment')
end
```
□

Instrukcje iteracyjne

Instrukcje iteracyjne służą do organizowania *cykli programowych*. Występują one w trzech odmianach, jako tzw. instrukcje *for*, *while* i *repeat*.

Instrukcja for

Instrukcja for składa się ze słowa kluczowego **for**, bezpośrednio po którym następuje identyfikator zmiennej sterującej cyklu, symbol przypisania, wyrażenie określające wartość początkową zmiennej sterującej, słowo kluczowe **to** albo **downto**, wyrażenie określające wartość końcową zmiennej sterującej, słowo kluczowe **do** oraz dowolna instrukcja.
Zmienna sterująca cyklu oraz oba wyrażenia muszą być tego samego typu porządkowego.

Składnia

instrukcja-for;

for *zmienna* := $wyrażenie_a$ **to** $wyrażenie_b$ **do** *instrukcja*

for *zmienna* := $wyrażenie_a$ **downto** $wyrażenie_b$ **do** *instrukcja*

Wykonanie instrukcji for powoduje wykonywanie zawartej w niej instrukcji dla tych wszystkich wartości zmiennej sterującej, które są zawarte w przedziale domkniętym wyznaczonym przez wartości $wyrażenia_a$ i $wyrażenia_b$.

Jeśli w instrukcji for wystąpiło słowo kluczowe **to**, to zmienna sterująca przybiera wartości od najmniejszej do największej. Jeśli natomiast wystąpiło słowo kluczowe **downto**, to przybiera ona wartości od największej do najmniejszej. Jeśli w pierwszym z tych dwóch przypadków jest prawdziwa relacja

$$wyrażenie_a > wyrażenie_b$$

albo gdy w drugim jest prawdziwa relacja

$$wyrażenie_a < wyrażenie_b$$

to cykl wyznaczony przez instrukcję for nie jest wykonywany w ogóle. W takim przypadku wykonanie instrukcji for ogranicza się do wyznaczenia wartości obu wymienionych wyrażeń.

W pozostałych przypadkach cykl jest wykonywany co najmniej jednokrotnie, a po jego zakończeniu zmienna sterująca cyklu ma wartość *wyrażenia*$_a$.

Pozostaje nadmienić, że wyznaczanie wartości omawianych wyrażeń jest jednokrotne, a jawne przypisywanie danych zmiennej sterującej cyklu jest zabronione.

Przykłady

```
for i := 2 to 8 do Writeln(i)

for j := 8 downto 2 do Writeln(j)

for toggle := false to true do
  for letter := 'i' to 'n' do
    arr[toggle,letter] := 2
```
□

Instrukcja while

Instrukcja while składa się ze słowa kluczowego **while**, bezpośrednio po którym następuje wyrażenie typu *boolean*, słowo kluczowe **do** oraz dowolna instrukcja.

Składnia

instrukcja-while:
 while *wyrażenie* **do** *instrukcja*

Wykonanie instrukcji while przebiega według poniższego algorytmu

1. Wyznaczana jest wartość wyrażenia.
2. Jeśli wartością tą jest *false*, to wykonanie instrukcji uznaje się za zakończone.
3. Jeśli wartością tą jest *true*, to jest wykonywana instrukcja następująca po słowie kluczowym **do**, a po tym następuje powtórzenie opisanych czynności od początku.

Przykłady

```
while i <> 0 do begin
  i := i - 1;
  Writeln(i)
end

while Flag do Fun(Flag)
```
□

Instrukcja repeat

Instrukcja repeat składa się ze słowa kluczowego **repeat**, bezpośrednio po którym następują: ciąg dowolnych instrukcji, słowo kluczowe **until** i wyrażenie typu *boolean*.

Składnia

instrukcja-repeat:
  repeat *instrukcje* **until** *wyrażenie*

Wykonanie instrukcji repeat przebiega według poniższego algorytmu

1. Wykonywane są instrukcje następujące po słowie kluczowym **repeat**.
2. Wyznaczana jest wartość wyrażenia występującego po słowie kluczowym **until**.
3. Jeśli wartością tą jest *true*, to wykonanie instrukcji uznaje się za zakończone.
4. Jeśli wartością tą jest *false*, to następuje powtórzenie opisanych czynności od początku.

Przykłady

```
repeat
  Writeln(i);
  i := i - 1
until i = 0;

repeat until KeyPressed
```
□

7. Typy porządkowe

Podstawowymi *typami danych* w Turbo Pascalu są *typy porządkowe*. Każdy z typów porządkowych ma tę właściwość, że definiujący go zbiór jest przeliczalny. Należy odnotować, że typ *real*, mimo iż jest typem prostym, nie jest typem porządkowym. To właśnie jest przyczyną, dla której wyrażenie selekcyjne instrukcji wyboru, etykiety instrukcji wyboru oraz zmienne sterujące i wyrażenia występujące w instrukcji *for* nie mogą być typu *real*.

Typy wyliczeniowe

Typy wyliczeniowe stanowią ważną odmianę typów porządkowych. Z każdym z nich jest związany zbiór o niewielkiej liczbie elementów, na których nie wykonuje się operacji arytmetycznych. Poszczególne elementy tych zbiorów są oznaczane unikalnymi identyfikatorami, a ich uporządkowanie wynika z kolejności wystąpienia tych identyfikatorów w opisie typu wyliczeniowego.

Opis typu wyliczeniowego składa się z listy identyfikatorów elementów tego typu, ujętej w nawiasy okrągłe. Każdy z takich identyfikatorów pełni rolę literału danego typu. Kolejność wystąpienia identyfikatorów na liście jest istotna.

Składnia

opis-typu-wyliczeniowego:
(*lista-identyfikatorów*)

Przykłady

type
Day = (*Mon,Tue,Wed,Thu,Fri,Sat,Sun*);
Color = (*Heart,Diamond,Spade,Club*);
RGB = (*Red,Green,Blue*);

• Dane typu *Day* są identyfikowane przez trzyliterowe nazwy symboliczne oznaczające dni tygodnia.
• Dane typu *Color* są identyfikowane przez nazwy symboliczne *Heart*, *Diamond*, *Spade* i *Club*.
• Dane typu *RGB* są identyfikowane przez nazwy symboliczne *Red*, *Green*, i *Blue*. □

W zasięgu opisu typu wyliczeniowego nie jest dozwolone użycie innego opisu typu wyliczeniowego, powołującego się na literały występujące w pierwszym z tych opisów.

Przykład

type
  Day = (*Mon*,*Tue*,*Wed*,*Thu*,*Fri*,*Sat*,*Sun*);
  WeekEnd = (*Sat*,*Sun*);

• Przytoczony zestaw definicji typu jest niepoprawny, ponieważ przecięcie list literałów typów *Day* i *WeekEnd* nie jest puste. □

Ponieważ typy wyliczeniowe są związane ze zbiorami przeliczalnymi, każdemu elementowi takiego zbioru można przypisać liczbę nieujemną, określającą miejsce elementu na liście identyfikatorów typu wyliczeniowego. W tym sensie dla definicji

type
  Direction = (*North*,*South*,*East*,*West*)

można przypisać identyfikatorowi *North* liczbę 0, identyfikatorowi *South* liczbę 1, identyfikatorowi *East* liczbę 2 i identyfikatorowi *West* liczbę 3. Dane o wartości tych liczb można uzyskać, posługując się predefiniowaną funkcją *ord*. Argumentem tej funkcji jest wyrażenie reprezentujące daną typu wyliczeniowego, a rezultatem nieujemna dana typu *integer*, której wartość określa liczony od 0 numer pozycji danej w zawierającym ją zbiorze uporządkowanym.

Dwiema innymi funkcjami, które mogą dotyczyć danych typu wyliczeniowego są *pred* i *succ*. Argumentem funkcji *pred* może być dowolne wyrażenie typu porządkowego, reprezentujące daną zbioru uporządkowanego. Rezultatem tej funkcji jest ta dana tego zbioru, która bezpośrednio poprzedza daną reprezentowaną przez argument. Zabrania się, aby argument funkcji *pred* reprezentował pierwszą daną zbioru, jako że ten jego element nie ma poprzednika.

Analogiczną do *pred* jest funkcja *succ*. Jej argumentem może także być dowolne wyrażenie typu wyliczeniowego, reprezentujące daną zbioru uporządkowanego. Rezultatem funkcji *succ* jest ta dana tego zbioru, która bezpośrednio następuje po danej reprezentowanej przez argument. Zabrania

się, aby argument funkcji *succ* reprezentował ostatnią daną zbioru, jako że ten jego element nie ma następnika.

Argumentami funkcji *ord*, *pred* i *succ* mogą być także wyrażenia reprezentujące dane typu *integer* i *byte*. W takim przypadku przyjmuje się, że jeśli użycie funkcji jest poprawne, to obowiązują następujące tożsamości

$$ord(n) = n$$
$$pred(n) = n - 1$$
$$succ(n) = n + 1$$

Przykłady
W zasięgu definicji predefiniowanego typu *boolean* oraz zdefiniowanego jawnie typu *Color*

```
type
  Color = (Heart,Diamond,Spade,Club);
```

jest prawdziwe następujące zestawienie:

Wyrażenie	Rezultat
ord (false)	0
pred (true)	*false*
ord (Club)	3
succ (Diamond)	*Spade*

Typy okrojone

Typy okrojone stanowią drugą, po typach wyliczeniowych, odmianę typów porządkowych. Z każdym z nich jest związany zbiór tych elementów wybranego zbioru uporządkowanego, których liczby porządkowe uzyskane za pomocą funkcji *ord* tworzą ciąg arytmetyczny o różnicy 1. Typ okrojony zachowuje wszystkie właściwości swego typu bazowego, z tym tylko wyjątkiem, że dane typu okrojonego muszą należeć do podzbioru stanowiącego okrojenie typu bazowego.

Opis typu okrojonego składa się z literału stanowiącego ograniczenie dolne zakresu typu okrojonego, symbolu .. (para kropek) oraz literału stanowiącego ograniczenie górne zakresu. Wymaga się, aby zakres wyznaczony przez wymienione tu literały nie był pusty.

Składnia

opis-typu-okrojonego:
  ograniczenie-dolne .. *ograniczenie-górne*
ograniczenie:
  literał
  nazwa-literału

Przykłady

type
Quadrant = 0..90;
Upper = '*A*'..'*Z*';
Logical = *false..true;*
Colour = (*Red,Blue,Green,Yellow,Orange*);
Hue = *Blue..Yellow*;

o Elementy zbioru danych typu *Quadrant* są reprezentowane przez liczby typu *integer* z przedziału od 0 do 90 włącznie.
o Elementy zbioru danych typu *Upper* są reprezentowane przez literały typu *char* z przedziału od '*A*' do '*Z*'.
o Elementy zbioru danych typu *Logical* są reprezentowane przez literały *false* i *true*, oba typu *boolean*.
o Elementy zbioru danych typu *Hue* są reprezentowane przez literały wyliczeniowe *Blue*, *Green* i *Yellow*, wszystkie typu *Colour*. □

Pozostaje nadmienić, że predefiniowany typ *byte* jest typem okrojonym typu *integer*, tj. że obowiązuje dla niego niejawna definicja

type
byte = 0..255;

W odniesieniu do danych typów okrojonych mogą być także stosowane funkcje *ord*, *pred* i *succ*. Wyznaczanie rezultatów tych funkcji odbywa się wówczas w typie bazowym.

Przykłady
W zasięgu definicji typu Hue

type
Colour = (*Red,Blue,Green,Yellow,Orange*);
Hue = *Blue..Yellow*;

jest prawdziwe następujące zestawienie:

Wyrażenie	Rezultat
ord(*Blue*)	1
succ(*Yellow*)	*Orange*
pred(*Yellow*)	*Green*

□

Należy zdecydowanie podkreślić, że posługiwanie się typami wyliczeniowymi i okrojonymi zwiększa czytelność programów, a ponadto ułatwia ich uruchamianie, jako że umożliwia automatyczne wykonywanie badań poprawności zakresu wartości danych. Nie bez znaczenia jest także fakt, że posługiwanie się danymi wymienionych typów prowadzi do oszczędności pamięci. Wynika to stąd, iż jeśli liczba elementów zbioru bazowego związanego z typem danej nie

przekracza 255, to Turbo Pascal reprezentuje daną w zaledwie jednym bajcie pamięci. Równie oszczędna reprezentacja jest przyjmowana także w odniesieniu do typów okrojonych z typem bazowym *integer*, jeśli oba ograniczenia zakresu mieszczą się w przedziale domkniętym [0 ; 255].

Konwersje typów

W Języku Turbo Pascal rozwinięto koncepcję funkcji *ord*, dokonującej *konwersji* danej dowolnego typu porządkowego w daną typu *integer*. Wprowadzono mianowicie rodzinę operatorów jednoargumentowych, o nazwach identycznych z nazwami typów porządkowych, umożliwiających wykonywanie konwersji danych między dowolnymi typami porządkowymi.

Operator konwersji ma postać *opr*(*arg*), gdzie *opr* jest identyfikatorem typu porządkowego, *arg* zaś jest dowolnym wyrażeniem porządkowym. Rezultatem takiej operacji jest dana typu *opr*, tak dobrana, że prawdziwa jest relacja *ord*(*a*) = *ord*(*opr*(*a*)).

Przykłady
W zasięgu definicji predefiniowanych typów porządkowych oraz zdefiniowanego jawnie typu *Color*

```
type
  Color = (Heart,Diamond,Spade,Club);
```

prawdziwe jest następujące zestawienie:

Wyrażenie	Rezultat
integer(*Club*)	3
Color(2)	*Spade*
char(65)	'*A*'
integer('*A*')	65
boolean(*Heart*)	*false*
byte(*true*)	1

□

Kontrola poprawności zakresu

Jedną z właściwości kompilatora języka Turbo Pascal jest możliwość takiego skompilowania programu źródłowego, aby podczas jego wykonywania były przeprowadzane kontrole poprawności zakresu danych skalarnych. Ponieważ kontrole takie spowalniają wykonywanie programu, są wstępnie wyłączone i mogą być włączane za pomocą dyrektywy {$R+}. W tych zakresach programu źródłowego, w których kontrole są zbędne, można je wyłączyć, posługując się dyrektywą {$R−}.

Przykład
Włączanie i wyłączanie kontroli zakresów danych

```
program Check;
type
  Figure = (Square,Circle,Diamond,Triangle);
var
  Shape : Figure;
begin
  Shape := Circle;
  Shape := Figure(4);
  {$R+}
  Shape := succ(Triangle);
  {$R-}
  Shape := pred(Square)
end.
```

- Kontrola poprawności zakresu jest włączona jedynie na czas wykonywania instrukcji przypisania zawartej między dyrektywami kompilacji.
- Wykonanie programu w podanej postaci spowoduje zasygnalizowanie błędu przekroczenia zakresu danych. Stanie się to podczas wyznaczania rezultatu funkcji *succ*.
- Gdyby z programu usunąć dyrektywy kompilacji, to zostałby on wykonany bez sygnalizowania błędów. □

8. Typy łańcuchowe

Każdy z *typów łańcuchowych* jest typem złożonym. Podczas definiowania typu łańcuchowego jest określana maksymalna liczba znaków, z których może składać się dana łańcuchowa tego typu. Z każdym typem łańcuchowym jest związany zbiór ciągów znaków. Należy do niego pusty ciąg znaków oraz ciągi o długości nie przekraczającej maksymalnej liczby znaków.

Opis typu łańcuchowego składa się ze słowa kluczowego **string**, bezpośrednio po którym następuje zawarta w nawiasach kwadratowych, maksymalna liczba znaków łańcucha danego typu. Liczba ta może być wyrażona za pomocą literału typu *integer* albo za pomocą równoważnej takiemu literałowi nazwy literału.

Składnia

opis-typu-łańcuchowego:
  string [*rozmiar*]
rozmiar:
  literal
  nazwa-literalu

Przykłady

type
  Cities = **string** [20];
  Names = **string** [12]; □

Każda z danych typu łańcuchowego zajmuje o jeden więcej bajtów pamięci niż wynosi maksymalna liczba znaków łańcucha danego typu. W tym dodatkowym bajcie jest przechowywany bieżący rozmiar łańcucha. Z tej informacji wynika, że maksymalny rozmiar łańcucha nie może przekroczyć 255 znaków.

Złączenia łańcuchów

Złączenie pary łańcuchów w jeden łańcuch może być zrealizowane za pomocą *operatora konkatenacji.* Operator ten jest zapisywany za pomocą znaku + (plus) i może dotyczyć pary wyrażeń łańcuchowych dowolnych typów. Złączenie pary łańcuchów jest łańcuchem składającym się z wszystkich znaków pierwszego łańcucha i następujących po nich wszystkich znaków drugiego łańcucha. Wymaga się, aby rozmiar łańcucha, który jest rezultatem złączenia, nie przekroczył 255 znaków.

Przykłady

Wyrażenie	Rezultat
'jan'+'ewa'	*'janewa'*
'5'+'.'+'4'	'5.4'
'j'+''+'b'	*'jb'*

(w ostatnim przykładzie wystąpił literał łańcuchowy reprezentujący pusty ciąg znaków). □

Relacje

Dowolne pary łańcuchów mogą być porównywane za pomocą *operatorów relacji.* Operatorami relacji są: = (równe), <> (nie równe), < (mniejsze), > (większe), <= (mniejsze lub równe), >= (większe lub równe). Priorytet każdego z operatorów relacji jest niższy niż priorytet operatora złączenia. Umożliwia to konstruowanie wyrażeń takich jak np. *'jan'* = *'j'*+*'an'* bez konieczności posługiwania się nawiasami.

Dwa łańcuchy są równe tylko wtedy, gdy są takiej samej długości oraz gdy składają się z identycznych znaków. Jeśli nie są równe, to relacja dotycząca łańcuchów zostaje zastąpiona relacją pierwszej pary odpowiadających sobie, ale różnych znaków. Jeśli para taka nie została znaleziona, ponieważ początkowym fragmentem jednego z łańcuchów jest drugi łańcuch, to za mniejszy zostanie uznany ten łańcuch, który jest krótszy.

Przykłady

Wyrażenie	Rezultat
'A' > *'B'*	*false*
'jan' > *'Jan'*	*true*
'' < char(0)	*true*
'' = ''	*true*
'Jan' <= *'Jane'*	*true*
'2599' < '270'	*true*

□

Przypisania

Przypisania łańcuchów zmiennym łańcuchowym mogą być realizowane za pomocą instrukcji przypisania. Z prawej strony dwuznaku przypisania może wystąpić dowolne wyrażenie łańcuchowe, natomiast z lewej nazwa zmiennej łańcuchowej. Jeśli długość łańcucha reprezentowanego przez wyrażenie przekracza maksymalny rozmiar łańcucha, który może być reprezentowany w zmiennej, to przypisanie odbywa się od lewej do prawej z pominięciem znaków nadmiarowych.

Przykład

```
type
  Name = string[7];
var
  FirstName,LastName : Name;
...
FirstName := 'Jan';
LastName := 'Bielecki';
```

• Zmiennej *FirstName* zostanie przypisana dana łańcuchowa o wartości literału '*Jan*'.

• Zmiennej łańcuchowej *LastName* zostanie przypisana dana łańcuchowa o wartości literału '*Bielecki*'. □

Podprogramy

Rozszerzeniem zbioru operacji łańcuchowych są *funkcje* i *podprogramy łańcuchowe*. Ich użycie znakomicie ułatwia przetwarzanie danych łańcuchowych.

Funkcja *Length*

Wywołanie: *Length*(*Str*)

Przyjmuje się, że *Str* jest wyrażeniem łańcuchowym reprezentującym ciąg znaków.

Rezultatem funkcji *Length* jest dana typu *integer* o wartości określającej liczbę znaków tego ciągu.

Przykłady

Wyrażenie	Rezultat
Length('*jb*')	2
Length('''')	1
Length('')	0

□

Procedura *Str*

Wywołanie: *Str*(*Val*,*Var*)

Przyjmuje się, że *Val* jest wyrażeniem arytmetycznym, a *Var* jest nazwą zmiennej łańcuchowej.

Wykonanie procedury *Str* powoduje przekształcenie danej reprezentowanej przez wyrażenie *Val* w ciąg wzorcowy mający postać literału o wartości tej danej, a następnie przekształcenie tego ciągu w daną łańcuchową i przypisanie zmiennej *Var*.

Jeśli wyrażenie *Val* jest typu *integer*, to przyjmuje się, że ciąg wzorcowy ma postać liczby całkowitej o minimalnej liczbie cyfr.

Jeśli bezpośrednio po wyrażeniu *Val* występuje kwalifikator o postaci :*m*, w którym *m* jest wyrażeniem typu *integer*, to określa on długość ciągu znaków, na który zostanie przekształcona dana reprezentowana przez *Val*. W ciągu takim wymieniony wyżej ciąg wzorcowy jest wyrównany prawostronnie. Jeśli ciąg wzorcowy nie da się przedstawić za pomocą *m* znaków, to *m* zostanie niejawnie zwiększone o tyle, aby było to wykonalne.

Przykłady
(Przyjęto, że *StrVar* jest typu **string**[4].)

Wywołanie	Ciąg	*StrVar*
Str(45,*StrVar*)	45	'45'
Str(−250,*StrVar*)	−250	'−250'
Str(45:3,*StrVar*)	*b*45	' 45'
Str(45:5,*StrVar*)	*bbb*45	' 4'
Str(−2:5,*StrVar*)	*bbb*−2	' −'

□

Jeśli wyrażenie *Val* jest typu *real*, to przyjmuje się, że ciąg wzorcowy ma postać

bsd.ddddddddddEsdd

w której *b* jest spacją, *s* jest znakiem mantysy lub wykładnika, a *d* jest cyfrą. Znak mantysy jest reprezentowany za pomocą spacji albo znaku − (minus), a znak wykładnika jest reprezentowany za pomocą znaku + (plus) albo − (minus).

Jeśli bezpośrednio po wyrażeniu *Val* występuje kwalifikator o postaci :*m*, w którym *m* jest wyrażeniem typu *integer*, to określa on długość ciągu znaków, na który zostanie przekształcona dana reprezentowana przez *Val*. W ciągu takim wymieniony wyżej literał zostanie wyrównany prawostronnie. Jeśli literał nie da się przedstawić za pomocą *m* znaków, to z wymienionego wyżej ciągu wzorcowego zostaną w pierwszej kolejności usunięte spacje wiodące, a jeśli i to nie poskutkuje, to zostanie odpowiednio zmniejszona liczba cyfr

mantysy, ale nie mniej niż do 2 cyfr. W przypadku redukowania liczby cyfr mantysy będzie stosowane zaokrąglenie.

Przykłady
(Przyjęto, że *Exp* ma wartość 45.678E1, a *StrVar* jest typu **string**[10].)

Wywołanie	Ciąg	*StrVar*
Str(*Exp*:9,*StrVar*)	4.568E+02	'4.568E+02'
Str(*Exp*:8,*StrVar*)	4.57E+02	'4.57E+02'
Str(*Exp*:7,*StrVar*)	4.6E+02	'4.6E+02'
Str(*Exp*:6,*StrVar*)	4.6E+02	'4.6E+02'
Str(*Exp*:10,*StrVar*)	4.5678E+02	'4.5678E+02'
Str(*Exp*:11,*StrVar*)	4.5678E+02	'4.5678E+0'

Jeśli bezpośrednio po *Val* występuje kwalifikator o postaci *:m:n*, w którym *m* i *n* są wyrażeniami typu *integer*, to omawiany literał jest wyrównywany prawostronnie w ciągu o długości *m* i niezależnie od wartości *m* jest przedstawiany jako zaokrąglony literał stałopozycyjny z *n* cyframi ułamkowymi. Jeśli literał nie da się przedstawić za pomocą *m* znaków, to *m* zostanie niejawnie zwiększone o tyle, aby było to wykonalne.

Przykłady
Przyjęto, że *Exp* ma wartość 45.678El, a *StrVar* jest typu **string**[10])

Wywołanie	Ciąg	*StrVar*
Str(*Exp*:6:2,*StrVar*)	456.78	'456.78'
Str(−*Exp*:6:2,*StrVar*)	−456.78	'−456.78'
Str(−*Exp*:8:2, *StrVar*)	*b*−456.78	' −456.78'
Str(−*Exp*:12:2,*StrVar*)	*bbbbb*−456.78	' −456.'
Str(−Exp:0:2,*StrVar*)	−456.78	'−456.'

□

Procedura *Val*

Wywołanie: *Val*(*Str*,*Var*,*Rep*)

Przyjmuje się, że *Str* jest wyrażeniem łańcuchowym, *Var* jest nazwą zmiennej typu *integer* albo *real*, a *Rep* jest nazwą zmiennej typu *integer*.

Wykonanie procedury *Val* powoduje potraktowanie znaków danej reprezentowanej przez *Str* jako liczby i przypisanie danej o wartości tej liczby zmiennej *Var*. Zabrania się, aby wspomniana liczba była poprzedzona spacjami, jak również zabrania się, aby po niej następowały spacje. Jeśli *Var* jest nazwą zmiennej typu *integer*, to wymaga się, aby także i liczba była typu *integer*.

Jeśli opisana konwersja jest wykonalna, to zmiennej *Rep* zostanie przypisana dana o wartości 0. W przeciwnym razie, zmiennej tej zostanie przypisana dana o wartości dodatniej. Wartość tej danej będzie stanowić numer tego znaku

danej łańcuchowej, który uniemożliwił dokonanie konwersji. W takim przypadku wartość zmiennej *Var* nie ulegnie zmianie.

Przykłady
(Przyjęto, że *IntVar* i *Rep* są nazwami zmiennych typu *integer*.)

Wywołanie	*IntVar*	*Rep*
Val('23',*IntVar*,*IntRep*)	23	0
Val('23.0',*IntVar*,*IntRep*)	b.z.	3
Val(' – 200',*IntVar*,*IntRep*)	b.z.	5

□

Funkcja *Copy*

Wywołanie: *Copy*(*Str*,*Pos*,*Num*)

Przyjmuje się, że *Str* jest wyrażeniem łańcuchowym, a *Pos* i *Num* są wyrażeniami typu integer.

Rezultatem funkcji *Copy* jest dana łańcuchowa składająca się z *Num* znaków łańcucha reprezentowanego przez *Str*, począwszy od znaku o numerze *Pos*. Jeśli *Pos* > *Length*(*Str*), to rezultatem funkcji jest dana łańcuchowa reprezentująca pusty ciąg znaków. Jeśli (*Pos* + *Num*) > *Length*(*Str*), to przyjmuje się *Num* := *Length*(*Str*) – *Pos* + 1. Zabrania się, aby *Pos* wykraczało poza zakres 1..255.

Przykłady

Wywołanie	Rezultat
Copy('*Branka*',3,4)	'*anka*'
Copy('*Branka*',5,3)	'*ka*'
Copy('*Branka*',7,2)	''
Copy('*Branka*',2,1)	'*r*'

□

Funkcja *Concat*

Wywołanie: *Concat*(*St1*,*St2*, ... ,*Stk*)

Przyjmuje się, że *St1*,*St2* ... ,*Stk* są wyrażeniami łańcuchowymi.

Rezultatem funkcji jest dana łańcuchowa *St1* + *St2* + ... + *Stk*. Zabrania się, aby suma długości argumentów przekraczała 255.

Przykłady

Wywołanie	Rezultat
Concat('*j*','*a*','*n*')	'*jan*'
Concat('*j*','','*b*')	'*jb*'
Concat('*jan*',' ','*b*')	'*jan b*'

□

Funkcja *Pos*

Wywołanie: *Pos(Src,Trg)*

Przyjmuje się, że *Src* i *Trg* są wyrażeniami łańcuchowymi.

Rezultatem funkcji *Pos* jest dana typu *integer*, której wartość określa najmniejszy numer tej pozycji w ciągu znaków reprezentowanym przez *Trg*, od której rozpoczyna się podciąg identyczny z podciągiem reprezentowanym przez *Src*. Jeśli podciąg taki nie występuje, to rezultatem funkcji jest dana o wartości 0.

Przykłady

Wywołanie	Rezultat
Pos('23','123123')	2
Pos('21','123123')	0
Pos('*c*','*abcabc*')	3
Pos('','*izabela*')	0

□

Procedura *Insert*

Wywołanie: *Insert(Src,Trg,Pos)*

Przyjmuje się, że *Src* jest wyrażeniem łańcuchowym, *Trg* jest nazwą zmiennej łańcuchowej, a *Pos* jest wyrażeniem całkowitym.

Wykonanie procedury *Insert* powoduje wstawienie do ciągu znaków reprezentowanego przez *Trg*, w miejscu przed jego znakiem określonym przez *Pos*, ciągu znaków reprezentowanego przez *Src*. Jeśli *Pos* > *Length*(*Trg*), to przyjmuje się, że *Pos* := *Length*(*Trg*) + 1. Zabrania się, aby *Pos* wykraczało poza zakres 1..255.

Jeśli wstawienie znaków spowodowałoby przekroczenie maksymalnego rozmiaru zmiennej *Trg*, to przypisanie jej danej reprezentującej nowy ciąg znaków odbędzie się od lewej do prawej, z odrzuceniem znaków nadmiarowych.

Przykłady
(Przyjęto, że zmienna *Trg* jest typu **string**[5], a przed każdym wykonaniem procedury *Insert* ma wartość '*jb*'.)

Wywołanie	*Trg*
Insert('*an*',*Trg*,2)	'*janb*'
Insert('*jb*',*Trg*,4)	'*jbjb*'
Insert('*janek*',*Trg*,1)	'*Janek*'

□

Procedura *Delete*

Wywołanie: *Delete(Str,Pos,Num)*

Przyjmuje się, że Str jest nazwą zmiennej łańcuchowej, a *Pos* i *Num* są wyrażeniami całkowitymi.

Wykonanie procedury *Delete* powoduje usunięcie z ciągu znaków reprezentowanego przez *Str*, porcji *Num* znaków, począwszy od pozycji *Pos*. Jeśli *Pos* > *Length*(*Str*), to wykonanie procedury nie ma żadnych skutków. Jeśli *Pos*+*Num*−1 > *Length*(*Str*), to przyjmuje się. że *Num*:= *Length*(*Str*) − *Pos*+1. Zabrania się, aby *Pos* wykraczało poza zakres 1..255.

Przykłady
(Przyjęto, że zmienna *Trg* jest typu **string**[5], a przed każdym wykonaniem procedury ma wartość *'janek'*.)

Wywołanie	*Trg*
Delete(*Trg*,4,2)	*'jan'*
Delete(*Trg*,6,1)	*'janek'*
Delete(*Trg*,5,3)	*'jane'*

□

Typy łańcuchowe i znakowe

Wszystkie typy łańcuchowe oraz predefiniowany typ porządkowy *char* są ze sobą zgodne. Oznacza to, że w tych wszystkich miejscach programu, w których jest poprawne użycie wyrażenia łańcuchowego, jest poprawne odwołanie do danej typu *char* i na odwrót. Wymaga się jedynie, aby podczas przypisywania wyrażenia łańcuchowego zmiennej typu *char* dana reprezentowana przez to wyrażenie była ciągiem znakowym o długości 1.

Dostęp do poszczególnych znaków zmiennej łańcuchowej można uzyskać nie tylko za pomocą predefiniowanej funkcji *Copy*, ale znacznie łatwiej za pomocą indeksowania. Jako definicję indeksowania można przyjąć równoważność

$$St[i] = Copy(St,i,1)$$

Równoważność ta jest jednak prawdziwa tylko wtedy, gdy zachodzi relacja $1 \leqslant i \leqslant Length(St)$. Poza tym zakresem możliwe jest sięganie do sąsiednich bajtów pamięci operacyjnej, a w szczególności posługiwanie się wyrażeniem *ord*(*St*[0]) w celu określenia bieżącej długości ciągu znakowego reprezentowanego przez *St*.

Należy nadmienić, że użycie dyrektywy kompilatora {$R+} tylko w niewielkim stopniu umożliwia wykrycie błędów związanych z niepoprawnym indeksowaniem. Wynika to stąd, że dyrektywa ta włącza kontrolę przekroczenia maksymalnego dopuszczalnego indeksu, znanego z deklaracji, ale nie wprowadza kontroli odwoływania się poza bieżący rozmiar zmiennej.

Przykład

```
var
  Name : string[5];
...
Name := 'Jan';
{$R+}
Name[0] := chr(5);
Name[4] := 'e';
Name[5] := 'k';
```

- Po wykonaniu przytoczonych przypisań zmienna *Name* ma wartość '*Janek*'.
- Użycie dyrektywy kompilatora {$R+} jest zbędne, ponieważ nie ma wpływu na przebieg wykonywania przypisań.
- Gdyby w zasięgu wspomnianej dyrektywy wystąpiło np. przypisanie

```
Name[6] := 'B'
```

to wykonywanie programu zostałoby wstrzymane na skutek przekroczenia maksymalnego dopuszczalnego zakresu indeksu. □

9. Typy tablicowe

Każdy z typów tablicowych jest typem złożonym. *Tablica* składa się z ustalonej liczby komponentów, z których każdy jest takiego samego typu. Komponenty mogą być zarówno typu prostego jak i typu złożonego. Dostęp do komponentów tablicy, nazywanych również *elementami*, odbywa się poprzez *indeksowanie*. *Indeksem* może być dowolne wyrażenie porządkowe ujęte w nawiasy kwadratowe. Dopuszczalny zakres indeksów jest określony w opisie typu tablicowego.

Opis typu tablicowego składa się ze słowa kluczowego **array**, bezpośrednio po którym następuje ujęty w nawiasy kwadratowe opis typu indeksów, słowo kluczowe **of**, a za nim opis typu elementów.

Składnia

opis-typu-tablicowego:
  array[*typ-indeksów*] **of** *typ-elementów*
typ-indeksów:
  opis-typu-porządkowego
typ-elementów:
  opis-typu

Przykłady

```
type
  Color = (Red,Green,Blue);
  TruthTable = array[boolean] of boolean;
  Height = 0..200;
var
  Square : array[Color] of Height;
  Matrix : array[2..8] of array[2..8] of integer;
  Table  : TruthTable;
```

- Typ *TruthTable* jest związany ze zbiorem tablic o indeksach i elementach typu *boolean*.
- Zmienna Square jest tablicą o indeksach typu *Color* i elementach typu *Height*.
- Zmienna *Matrix* jest tablicą o indeksach typu 2..8 i elementach, które są tablicami o indeksach typu 2..8 i elementach typu *integer*.
- Zmienna *Table* jest tablicą typu *TruthTable*, tj. tablicą o indeksach i elementach typu *boolean*. □

Elementy tablic są reprezentowane w programach za pomocą tzw. *nazw elementów tablic*. Nazwa elementu tablicy składa się z nazwy tablicy, bezpośrednio po której następuje wyrażenie indeksowe ujęte w nawiasy kwadratowe. Wyrażenie indeksowe może być dowolnym wyrażeniem porządkowym takiego samego typu jaki pod nazwą *typ-indeksów* występuje w opisie typu tablicowego.

Użycie dyrektywy {$*R*+} powoduje generowanie kodu dokonującego kontroli poprawności wyrażeń indeksowych. Kontrola ta polega na upewnieniu się, że wartość wyrażenia indeksowego mieści się w zakresie wyznaczonym przez typ indeksów w opisie typu tablicowego.

Przykład

```
type
  Color = (Red,Green,Blue);
var
  Matrix : array[Color] of boolean;
```

- Zmienna *Matrix* jest tablicą o indeksach typu *Color* i elementach typu *boolean*.
- Poprawnymi nazwami elementów tablicy *Matrix* są m. in.

```
Matrix[Green]
Matrix[pred(Blue)]
Matrix[Color(ord(true))]
```

- W zasięgu dyrektywy kompilatora {$*R*+} taka nazwa elementu tablicy jak np. *Matrix*[*succ*(*Blue*)] zostałaby rozpoznana jako niepoprawna. □

Tablice wielowymiarowe

Ponieważ elementami tablic mogą być tablice, łatwo jest konstruować tablice dwu- i więcejwymiarowe. Zgodnie ze składnią, definicja typu opisującego tablicę dwuwymiarową przybiera np. postać

```
type
  TwoDim = array[2..5] of array[boolean] of real;
```

Definicja taka może być zapisana w sposób uproszczony

type
TwoDim = **array**[2..5,*boolean*] **of** *real*;

tzn. opisy typu indeksów mogą być ujęte w listę zawartą w nawiasach kwadratowych.

Podobne uproszczenie może być zastosowane również w odniesieniu do nazw elementów tablic. W zasięgu deklaracji takiej jak np.

var
Matrix : *TwoDim*;

za poprawne i równoważne uznaje się bowiem takie m.in. nazwy elementów tablicy *Matrix* jak *Matrix*[3] [*true*] i *Matrix*[3,*true*].

Przykład

type
Pupils = **string**[20];
Class = **array**[1..30] **of** *Pupils*;
var
School : **array**[1..20] **of** *Class*;

- Typ Class jest związany ze zbiorem, którego elementami są tablice zawierające wykazy nazwisk uczniów pewnej klasy.
- Zmienna *School* jest tablicą dwuwymiarową, typu

array[1..20] **of** **array**[1..30] **of** **string**[20]

w której przechowywane są dane określające nazwiska uczniów każdej z 20 klas danej szkoły.
- Nazwa *School*[3] [25] dotyczy ucznia klasy nr 3, występującego na liście uczniów tej klasy na pozycji 25. Nazwa ta jest równoważna nazwie uproszczonej *School*[3,25]. □

Wektory znakowe

Wektorami znakowymi są tablice jednowymiarowe, o typie indeksów *integer*, których elementy są typu standardowego *char*. Wektory takie mogą być uważane za zmienne łańcuchowe reprezentujące ciągi znaków o stałej długości. Dzięki takiej interpretacji nazwy wektorów znakowych i nazwy ich elementów mogą występować w wyrażeniach łańcuchowych wszędzie tam, gdzie mogą występować nazwy zmiennych łańcuchowych. Dotyczy to w szczególności relacji.

Przykład

```
type
  Number = 2..5;
  StrTyp = string[5];
  ArrTyp = array[Number] of char;
var
  StrVar : StrTyp;
  ArrVar : ArrTyp;
  i : byte;
begin
  for i:= 2 to 5 do
    ArrVar[i]:= ord(63+i);
  StrVar:= '98765';
  StrVar:= Copy(StrVar,2,2)+ArrVar+'/'+ArrVar[4];
end.
```

- Po wykonaniu drugiej instrukcji przypisania zmienna *StrVar* ma wartość '87*ABCD/C*'.
- W tym samym momencie relacja

ArrVar > *StrVar*

ma wartość *true*. □

Przypisania

Zgodnie z zasadą, że zmiennej dowolnego typu może być przypisana dana takiego samego typu, jest możliwe przypisanie, za pomocą jednej instrukcji, całej tablicy danej tablicowej. Dozwolone jest również przypisanie tablicy znakowej danej reprezentowanej przez literał znakowy, ale tylko wtedy, gdy liczba znaków literału jest równa liczbie znaków tablicy.

Przykład

```
type
  Line = string[60];
  Page = array[1..30] of Line;
  Chapter = array[1..50] of Page;
  Word = array[1..5] of char;
var
  Book1,Book2 : array[1..6] of Chapter;
  Arr : Word;
  Brr : array[1..5] of char;
  Str : string[5];
```

o W zasięgu przytoczonych definicji i deklaracji poprawne są m.in. następujące przypisania:

Book1 := *Book2*
Arr := *'Janek'*
Str := *Arr*

o Niepoprawne są natomiast przypisania

Arr := *'Jan'* – ponieważ literał liczy za mało znaków
Arr := *Str* – ponieważ *Str* jest wyrażeniem łańcuchowym
Arr := *Brr* – ponieważ *Arr* jest typu *Word*, a *Brr* nie jest typu *Word*. □

Tablice predefiniowane

W języku Turbo Pascal predefiniowano dwie tablice o elementach typu *byte* i dwie tablice o elementach typu *byte*, umożliwiające dostęp do pamięci operacyjnej oraz do portów wejścia/wyjścia.

Tablice *Mem* i *MemW*

Tablice *Mem* i *MemW* zapewniają dostęp do pamięci operacyjnej. Pierwsza z nich udostępnia bajt, a druga – słowo. Indeksem w każdym odwołaniu do tych tablic jest wyrażenie adresowe składające się z oddzielonych znakiem : (dwukropek) dwóch wyrażeń typu *integer*, określających kolejno: numer segmentu i adres względny w segmencie pamięci.

Przykład

ByteVar := *Mem*[0000 : 0023]
MemW[*Seg*(*WordVar*) : *Ofs*(*WordVar*)] := $*FFAA*

o Wykonanie pierwszej instrukcji powoduje przypisanie zmiennej *ByteVar* danej reprezentowanej w bajcie pamięci operacyjnej znajdującym się w segmencie 0000 pod adresem względnym 0023.

o Wykonanie drugiej instrukcji powoduje umieszczenie danej słowowej o wartości $*FFAA* w pierwszych dwóch bajtach zmiennej *WordVar*. Predefiniowane funkcje *Seg* i *Ofs* zostały tu wykorzystane do określenia segmentu i adresu względnego w segmencie, przypisanych pierwszemu bajtowi pamięci zajętemu przez zmienną *WordVar*. □

Tablice *Port* i *PortW*

Tablice *Port* i *PortW* zapewniają dostęp do portów wejścia/wyjścia. Typem indeksów tych tablic jest *integer*. Elementy tablic *Port* i *PortW* nie mogą

występować jako argumenty odwołań do podprogramów, ale mogą występować w wyrażeniach oraz z lewej strony symbolu przypisania. Jeśli występują w wyrażeniu, to reprezentują daną pochodzącą z podanego portu. Jeśli występują z lewej strony symbolu przypisania, to następuje przesłanie danej do podanego portu.

Przykład

Port[34] := $5*F*;
WordVar := *PortW*[34];

- Wykonanie pierwszej instrukcji powoduje umieszczenie w porcie 8-bitowym o adresie 34 danej reprezentowanej przez literał $5*F*.
- Wykonanie drugiej instrukcji powoduje sięgnięcie do portu 16-bitowego i przypisanie znajdującej się tam danej, zmiennej *WordVar*. □

10. Typy rekordowe

Każdy z typów rekordowych jest typem złożonym. *Rekord* składa się z ustalonej liczby komponentów, które mogą być różnych typów. Komponenty mogą być zarówno typu prostego jak i złożonego. Dostęp do komponentów, nazywanych również *polami*, odbywa się poprzez *selekcję*. Selekcja polega na określeniu nazwy rekordu, po której następuje znak . (kropka) i identyfikator pola rekordu.

Opis typu rekordowego składa się ze słowa kluczowego **rekord**, bezpośrednio po którym następuje wykaz pól rekordu i słowo kluczowe **end**. Każdy element wykazu ma postać deklaracji pola. Deklaracje pól są oddzielone średnikami, a ostatnim elementem wykazu może być *część wariantowa rekordu.* Część wariantowa składa się ze słowa kluczowego **case**, bezpośrednio po którym następuje deklaracja pola wyróżnikowego albo opis typu wyróżnikowego, słowo kluczowe **of**, a bezpośrednio po nim wykaz wariantów. Każdy element wykazu wariantów składa się z listy etykiet wyboru zakończonej dwukropkiem i ujętego w nawiasy okrągłe wykazu pól wariantu. Poszczególne elementy wykazu wariantów są oddzielone średnikami.

Składnia

opis-typu-rekordowego
record *wykaz-deklaracji-pól-rekordu* **end**
wykaz-deklaracji-pól-rekordu;
deklaracja-pola
część-wariantowa-rekordu
deklaracja-pola:
lista-nazw-pól : *opis-typu*
nazwa-pola:
identyfikator

część-wariantowa-rekordu:
  case *deklaracja-pola-wyróżnikowego* **of** *wykaz-wariantów*
  case *opis-typu-wyróżnikowego* **of** *wykaz-wariantów*
deklaracja-pola-wyróżnikowego:
  nazwa-pola-wyróżnikowego : *opis-typu-wyróżnikowego*
nazwa-pola-wyróżnikowego:
  identyfikator
opis-typu-wyróżnikowego:
  identyfikator-typu-porządkowego
wykaz-wariantów:
  lista-etykiet-wyboru : (*wykaz-pól-wariantu*)
etykieta-wyboru:
  literał
  nazwa-literału

Przykłady

```
type
  Days = 1..31;
  Date = record
           Day : Days;
           Month : 1..12;
           Year : 1900..1999
         end;
  Person = record
             Name : string[20];
             BirthPlace : string[30];
             case Sex : (male,female,alien) of
               male,female : (BirthDate : Date);
               alien : (BirthDate : real;
                        BodyData : record
                                     H : integer;
                                     W : real
                                   end)
           end;
var
  MyBirthDay : Date;
  EarthMan,Galaxian : Person;
```

- Typ *Date* jest związany ze zbiorem rekordów, których pola są typu *Days*, 1..12 i 1900..1999.
- Typ *Person* jest związany ze zbiorem rekordów z wariantami. Wyróżnikiem wariantów jest pole *Sex* typu (*male,female,alien*). Jeśli wyróżnik *Sex* ma wartość *male* albo *female*, to rekord typu *Person* składa się z pól typu

string[20], **string**[30], (*male,female,alien*), 1..31, 1..12 i 1900..1999. Jeśli wyróżnik *Sex* ma wartość *alien*, to rekord typu *Person* składa się z pól typu **string**[20], **string**[30], (*male,female,alien*), *real*, *integer* i *real*.

• Zmienna *MyBirthDay* jest rekordem składającym się z pól *Day*, *Month* i *Year*.

• Zmienne *EarthMan* i *Galaxian* są rekordami typu *Person*. Wybór wariantu rekordów odbywa się przez przypisanie polu wyróżnikowemu, danej typu (*male,female,alien*).

• Pole wyróżnikowe zmiennej *Earthman* jest identyfikowane przez selekcję *EarthMan.Sex*, a pole wyróżnikowe zmiennej *Galaxian* przez selekcję *Galaxian.Sex*.

• Po wykonaniu przypisania

Galaxian.Sex := *alien*

jest dozwolone odwoływanie się do komponentów wariantu *alien*. W szczególności poprawne jest wówczas przypisanie

Galaxian.BodyData.H := 20

w którym operator selekcji . (kropka) zastosowano najpierw do komponentów rekordu *Galaxian*, a następnie do komponentów rekordu *Galaxian.Body Data*. □

Instrukcja with

Odwoływanie się do komponentów rekordu wymaga podania nie tylko nazwy pola, ale również nazwy samego rekordu. Ponieważ w pewnych przypadkach znacznie wydłuża to tekst programu, wygodnie jest posłużyć się *instrukcją wiążącą* with „odsłaniającą" wnętrze definicji typu rekordowego.

W ogólnym przypadku instrukcja with składa się ze słowa kluczowego **with**, bezpośrednio po którym następuje lista nazw rekordów, słowo kluczowe **do** i dowolna instrukcja. W jej obrębie selekcja pola rekordu wymienionego na liście może być zastąpiona samą nazwą pola. Przyjmuje się, z definicji, że instrukcja o postaci

with *a*, *b*, ... , *z* **do** *i*

jest równoważna instrukcji

```
with a do
  with b do
    ...
      with z do i
```

Składnia

instrukcja with:

with *lista-nazw-rekordów* **do** *instrukcja*

nazwa-rekordu:

identyfikator

nazwa-elementu-tablicy

nazwa-pola

Przykład

```
var
  Alfa : record
           Beta : record
                    Gamma : char;
                    Delta : real
                  end;
           Gamma : integer
         end;
  Beta : array[boolean] of record
                             Chi,Tau : real
                           end;
```

• W zasięgu przytoczonej deklaracji jest poprawna instrukcja

```
with Alfa.Beta,Beta[true] do
  Delta := Tau
```

wykonywana tak jak instrukcja

```
Alfa.Beta.Delta := Beta[true].Tau
```

• Poprawna jest także instrukcja

```
with Alfa.Beta,Alfa do begin
  Delta := 23.4;
  Writeln(Gamma)
end
```

wykonywana jak instrukcja

```
begin
  Alfa.Beta.Delta := 23.4;
  Writeln(Alfa.Gamma)
end
```

• Należy zwrócić uwagę, że kolejność nazw rekordów na liście instrukcji with jest istotna, gdyż instrukcja taka jak

```
with Alfa,Alfa.Beta do begin
  Delta := 23.4;
  Writeln(Gamma)
```

```
    end
```

jest wykonywana tak jak instrukcja

```
begin
  Alfa.Beta.Delta := 23.4;
  Writeln(Alfa.Beta.Gamma)
end
```

a nie jak instrukcja

```
begin
  Alfa.Beta.Delta := 23.4;
  Writeln(Alfa.Gamma)
end
```

□

Traktowanie wariantów jako unii

Jeśli rekord zawiera część wariantową, to w reprezentacji wewnętrznej przyjętej w Turbo Pascalu, pierwsze pole każdego wariantu zaczyna się od tego samego bajtu pamięci. Biorąc ponadto pod uwagę, że deklaracja pola wyróżnikowego części wariantowej może zostać uproszczona do opisu typu, łatwo jest przekształcić opis rekordu w opis *unii*. Należy jedynie uwzględnić, iż w odróżnieniu od operacji na rekordach operacje na uniach wymagają znajomości sposobu reprezentowania komponentów, a przynajmniej sposobu rozmieszczenia i rozmiaru każdego z nich.

Przykład

```
var
  Word : record
           case boolean of
             false : (LowByte,HighByte : byte);
             true : (FullWord : integer)
         end
```

• W zasięgu przytoczonej deklaracji jest dopuszczalne zastąpienie instrukcji

```
with Word do
  FullWord := FullWord and $FF00
```

instrukcją

```
with Word do
  LowByte := 0
```

• Poprawność przytoczonego postępowania wynika z faktu, że w implementacji dla IBM PC mniej znaczący bajt danej typu integer zajmuje w pamięci operacyjnej bajt o niższym adresie, a pola struktury występują w pamięci operacyjnej w takiej samej kolejności w jakiej zostały zadeklarowane (tu *LowByte* przed *HighByte*). □

Przypisania rekordów

Analogicznie do przypisywania zmiennym tablicowym całych tablic jest dozwolone przypisywanie zmiennym rekordowych całych rekordów. Wymaga się jedynie, aby typ zmiennej rekordowej był identyczny z typem przypisywanego jej rekordu.

Przykład

```
type
  Name = record
             LastName : string[32];
             Names : array[(fst,snd,trd)] of string[8]
           end;
var
  JohnName,EwaName : Name;
...
EwaName:= JohnName;
```



11. Typy mnogościowe

Każdy z typów mnogościowych jest typem złożonym. *Mnogość* jest zmienną, której może być przypisany wybrany podzbiór pewnego zbioru potęgowego. Zbiorem bazowym zbioru potęgowego może być dowolny zbiór elementów typu porządkowego. Każdy element mnogości jest jednym z takich właśnie elementów.

Dwie dane mnogościowe są uznawane za równe wtedy i tylko wtedy, gdy składają się z takich samych elementów. Jeśli wszystkie elementy jednej mnogości są elementami drugiej, to mówi się, że pierwsza mnogość zawiera się w drugiej.

Opis typu mnogościowego składa się ze słowa kluczowego **set**, bezpośrednio po którym następuje kluczowe **of** i opis typu bazowego. Wymaga się, aby typ porządkowy pełniący rolę typu bazowego nie zawierał więcej niż 256 elementów oraz aby rezultat funkcji *ord* dla każdego z tych elementów mieścił się w przedziale domkniętym [0 ; 255].

Składnia

opis-typu-mnogościowego
set of *typ-bazowy*
typ-podstawowy:
typ-porządkowy

Przykłady

type
DaysOfWeek = **set of** (*Mon,Tue,Wed,Thu,Fri,Sat,Sun*);
Chars = **set of** *char*;
SmallLetters = **set of** '*a*'..'*z*';
DayNumbers = **set of** 1..31;

- Wartością danej typu *DaysOfWeek* może być dowolny, w tym pusty, podzbiór zbioru nazw dni tygodnia.
- Wartością danej typu *Chars* może być dowolny podzbiór zbioru znaków kodu ASCII.
- Wartością danej typu *SmallLetters* może być dowolny podzbiór zbioru małych liter alfabetu ASCII.
- Wartością danej typu *DayNumbers* może być dowolny podzbiór zbioru liczb całkowitych z przedziału domkniętego [1, 31]. □

Literały mnogościowe

Literał mnogościowy składa się z zawartej w nawiasach kwadratowych listy literałów, z których każdy jest tego samego typu porządkowego. Lista ta może być pusta, może zawierać powtórzenia elementów oraz może zawierać konstrukcje o postaci *min..max*, w których *min* i *max* są literałami tego samego typu. W tym ostatnim przypadku napis *min..max* jest równoważny liście wszystkich literałów od *min* do *max* włącznie. Jeśli *min* > *max*, to napis ten opisuje podzbiór pusty.

Należy nadmienić, że wartość literału mnogościowego nie zależy od kolejności tworzących go literałów typu bazowego, a wielokrotne ich wystąpienia nie mają wpływu na wartość literału mnogościowego.

Przykład

Kilka poprawnych i niepoprawnych literałów mnogościowych

Literały poprawne:	['*P*','*a*','*s*','*c*','*a*','*l*']
	[4,9,5,0,8,6]
	[*true*,*false*]
	[ ]
	[20..30,50,70..80]

Napisy, które nie są literałami mnogościowymi.

[*true*,2]	– różne typy bazowe
[200..300]	– *ord*(300) > 255
[4.0,5.0]	– typ bazowy nie jest typem porządkowym

□

Wyrażenia

Wyrażenia mnogościowe składają się z odwołań do zmiennych mnogościowych, odwołań do literałów i konstruktorów mnogościowych oraz operatorów sumy, różnicy i przecięcia.

Konstruktory mnogościowe mają postać zbliżoną do literałów mnogościowych, tyle że elementami ich list mogą być nie tylko literały typu podstawowego, ale również dowolne wyrażenia tego typu.

Operator sumy mnogości jest zapisywany za pomocą znaku + (plus), *operator różnicy* za pomocą znaku – (minus), a *operator przecięcia* za pomocą znaku * (gwiazdka).

Sumą mnogości jest mnogość składająca się z wszystkich elementów obu składników. *Różnicą mnogości* jest mnogość składająca się z tych wszystkich elementów odjemnej, które nie są elementami odjemnika. *Przecięciem mnogości* jest mnogość składająca się z wszystkich elementów wspólnych czynników przecięcia.

Poza operacjami mnogościowymi, dane typu mnogościowego mogą występować w relacjach

a = *b*	prawdziwej, jeśli *a* i *b* są zbiorami tych samych elementów.
a <> *b*	prawdziwej, jeśli *a* i *b* są różnymi zbiorami elementów.
a <= *b*	prawdziwej, jeśli *a* jest pustym zbiorem elementów albo gdy każdy element należący do *a* należy także do *b*.
a >= *b*	prawdziwej, gdy *a* jest pustym zbiorem elementów albo gdy każdy element należący do *b* należy także do *a*.
e **in** *a*	prawdziwej, gdy element typu podstawowego *e* jest elementem zbioru elementów *a*.

Przykłady

W zasięgu deklaracji zmiennej *SetVar*, której przypisano daną mnogościową reprezentowaną przez literał [*Red*,*Green*].

```
type
  Colors = (Red,Green,Blue);
  Paint = set of Colors;
var
  SetVar : Paint;
...
SetVar := [Red,Green];
```

jest prawdziwe następujące zestawienie:

Wyrażenie	Rezultat
SetVar = [*Red*..*Blue*] – [*Blue*]	*true*
SetVar <> []	*true*
SetVar <= [*Red*..*succ*(*Green*)]	*true*
SetVar >= [*Blue*..*Green*]	*true*
SetVar * [*Red*] = [*Red*]	*true*
Green **in** *SetVar*	*true*

□

Przypisania mnogości

Analogicznie do przypisania zmiennym tablicowym całych tablic oraz zmiennym rekordowym całych rekordów, jest dozwolone przypisywanie zmiennym mnogościowym mnogości. Wymaga się jedynie, aby typ zmiennej mnogościowej był identyczny z typem przypisywanej jej danej mnogościowej.

Przykład

```
var
  SetVar : array[2..5, boolean] of set of 20..30;
...
SetVar[3,false] := [5*5..pred(29)];
```

• Prawą stronę instrukcji przypisania stanowi konstruktor mnogościowy. Konstruktor ten jest równoważny literałowi mnogościowemu [25,26,27,28]. □

12. Typy plikowe

Każdy z typów plikowych jest typem złożonym. *Plik* składa się z komponentów, z których każdy jest takiego samego typu. Komponenty pliku są nazywane jego elementami. W odróżnieniu od typu tablicowego liczba elementów pliku nie jest ustalona w deklaracji pliku, lecz jest uzależniona od przebiegu wykonywania programu, a w szczególności od tego, z jakim zbiorem danych rozpatrywany plik zostanie skojarzony.

Należy podkreślić, że obiekt nazywany plikiem jest jedynie abstrakcyjnym modelem fizycznego zbioru danych występującego najczęściej poza programem. Fizyczne zbiory danych — nazywane tu krótko *zbiorami* — mogą rezydować w pamięci zewnętrznej komputera, mogą zajmować część jego pamięci operacyjnej, a także mogą być utożsamiane ze strumieniami danych wprowadzanych i wyprowadzanych za pomocą urządzeń zewnętrznych.

Możliwość przetwarzania różnych zbiorów danych za pomocą tego samego pliku, po uprzednim skojarzeniu pliku ze zbiorem, stanowi znaczne ułatwienie programowania, ponieważ ogranicza zadanie programującego do koncentrowania się na istotnych elementach algorytmu przetwarzania bez wnikania w działania specyficzne dla danego systemu operacyjnego oraz w sposób reprezentowania danych w pamięci zewnętrznej. Z tego względu programowanie na poziomie abstrakcji nazywanej plikiem sprowadza się głównie do otwarcia pliku, wykonania operacji na jego elementach oraz do zamknięcia pliku.

Opis typu plikowego składa się ze słowa kluczowego **file**, bezpośrednio po którym następuje słowo kluczowe **of**, a za nim opis typu elementów pliku. Elementami pliku mogą być obiekty dowolnych typów prostych, a także dowolne agregaty z wyjątkiem plików.

Składnia

opis-typu-plikowego:
file of *typ-elementów-pliku*
typ-elementów-pliku:
opis-typu

Przykłady

```
type
  Measurements = file of real;
  Persons = file of
              record
                Name : string[20];
                Sex : (male,female);
                Age : 18..65
              end;
var
  Experiment : Measurements;
  Population : Persons;
  Complexes : file of record
                        Re,Im : real
                      end;
  Arrays : file of array[2..6,4..8,boolean] of byte;
```

- Typ *Measurements* jest związany ze zbiorem sekwencji danych typu *real*.
- Typ *Persons* jest związany ze zbiorem sekwencji składających się z rekordów o komponentach typu **string**[20], (*male,female*) i 18..65.
- Zmienna *Experiment* jest zmienną plikową typu *Measurement*.
- *Complexes* jest zmienną plikową reprezentującą plik związany ze zbiorem sekwencji par danych typu *real*.
- Zmienna *Arrays* jest zmienną plikową reprezentującą plik związany ze zbiorem sekwencji tablic typu **array**[2..6,4..8,*boolean*] **of** *byte*. □

Podprogramy

Operacje na plikach mogą być wykonywane jedynie za pomocą wywołań funkcji i procedur. Każda z takich operacji musi być poprzedzona skojarzeniem pliku ze zbiorem danych, a jeśli dotyczy elementów pliku, to musi być także poprzedzona otwarciem pliku. Skojarzenie pliku ze zbiorem odbywa się za pomocą procedury *Assign*, a otwarcie pliku za pomocą procedur *Reset* i *Rewrite*. Użycie procedury *Reset* nie implikuje, że plik jest otwierany wyłącznie w celu wprowadzania elementów zbioru, a użycie procedury *Rewrite*

nie implikuje, że plik jest otwierany wyłącznie w celu ich wyprowadzania. Przed zakończeniem wykonywania programu każdy plik powinien zostać zamknięty. Służy do tego procedura *Close*. Samo zakończenie wykonywania programu nie pociąga za sobą zamknięcia nie zamkniętych jeszcze plików.

Bezpośrednio po otwarciu plik znajduje się w pozycji początkowej. Zmiana pozycji pliku może być uzyskana za pomocą procedury *Seek*. Aktualny rozmiar pliku może być określony za pomocą funkcji *FileSize*, a jego aktualna pozycja za pomocą funkcji *FilePos*. Jeśli plik zostanie ustawiony w pozycji pośredniej, tj. między pozycją początkową i końcową, to każde wykonanie procedury *Write* spowoduje zmianę najbliższego elementu pliku. Zmiana ta nie będzie miała żadnego wpływu na stan pozostałych elementów pliku. Z tego względu pliki o elementach ustalonego typu mogą reprezentować zbiory o organizacji sekwencyjno-wyrywkowej.

Procedura *Assign*

Wywołanie: *Assign*(*FileVar*,*StrExp*)

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej, a *StrExp* jest wyrażeniem łańcuchowym. Wymaga się, aby plik *FileVar* nie był otwarty.

Wykonanie procedury *Assign* powoduje skojarzenie pliku *FileVar* ze zbiorem danych o nazwie określonej przez wyrażenie *StrExp*.

Przykład

```
var
  Results:file of real;
...
Assign(Results,'A:TESTS.DOC');
```

- Zmienna plikowa *Results* reprezentuje plik o elementach typu *real*.
- Wykonanie procedury *Assign* powoduje skojarzenie pliku *Results* ze zbiorem danych *TESTS.DOC* znajdującym się w domniemanym katalogu stacji dyskowej *A*.
- Wykonanie procedury *Assign* nie powoduje otwarcia pliku *Results*. □

Procedura *Reset*

Wywołanie: *Reset*(*FileVar*)

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej. Wymaga się, aby przed wywołaniem procedury Reset plik *FileVar* był skojarzony z istniejącym zbiorem danych.

Wykonanie procedury *Reset* powoduje otwarcie pliku *FileVar*. Bezpośrednio

po otwarciu plik zostaje ustawiony w pozycji początkowej, tj. tuż przed jego pierwszym elementem.

Przykład

var
  InpFile : **file of record**
    Name : **string**[30];
    Income : *real*
   end;

...

Assign(*InpFile*,'*INCOME.DOC*');
Reset(*InpFile*);

• Wykonanie procedury *Assign* powoduje skojarzenie pliku *InpFile* ze zbiorem *INCOME.DOC* znajdującym się w domniemanym katalogu domniemanej stacji dyskowej.
• Wykonanie procedury *Reset* powoduje otwarcie pliku *InpFile*.
• Otwarcie pliku za pomocą *Reset* nie wyklucza możliwości przyszłego odwoływania się do niego za pomocą procedur *Seek* i *Write*. □

Procedura *Rewrite*

Wywołanie: *Rewrite*(*FileVar*)

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej. Wymaga się, aby przed wywołaniem procedury *Rewrite* plik *FileVar* był skojarzony ze zbiorem danych.

Wykonanie procedury *Rewrite* powoduje otwarcie pliku *FileVar*. Bezpośrednio po otwarciu plik zostaje ustawiony w pozycji początkowej, tj. tuż przed jego pierwszym elementem.

Jeśli przed wykonaniem procedury *Rewrite* zbiór skojarzony z plikiem nie istniał, to zostanie utworzony. Jeśli istniał, to zostanie usunięty i utworzony ponownie. W obu przypadkach zostanie utworzony zbiór pusty.

Przykład

var
  OutFile : **file of array** [1..20] **of** *byte*;

...

Assign(*OutFile*,'*B*: *TESTS.OUT*');
Rewrite(*OutFile*);

• Wykonanie procedury *Assign* powoduje skojarzenie pliku *OutFile* ze zbiorem *TESTS.OUT* znajdującym się na dyskietce umieszczonej w stacji dyskowej *B*.

• Wykonanie procedury *Rewrite* powoduje otwarcie pliku *OutFile*. Po dokonaniu otwarcia plik ten jest pusty.
• Otwarcie pliku za pomocą procedury *Rewrite* nie wyklucza możliwości przyszłego odwoływania się do niego za pomocą wywołań procedur *Seek* i *Read*. □

Procedura *Read*

Wywołanie: *Read(FileVar,VarList)*

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej, a *VarList* jest listą nazw zmiennych. Wymaga się, aby plik *FileVar* był otwarty. Zabrania się, aby jakakolwiek zmienna listy *VarList* była innego typu niż typ elementów pliku.

Wykonanie procedury *Read* z listą nazw zmiennych jest równoważne wykonaniu ciągu procedur *Read* z nazwami tych zmiennych. Wykonanie procedury *Read* z nazwą zmiennej powoduje wprowadzenie z pliku jednego elementu i przypisanie go zmiennej o podanej nazwie.

Przykład

```
type
  ElmType = record
              Re,Im : real
            end
var
  InpFile : file of ElmType;
  ArrVar : array[boolean,2..4] of ElmType;
...
Assign(InpFile,'COMPLEX.DOC');
Reset(InpFile);
Read(InpFile,ArrVar[true,3]);
```

• Wykonanie procedury *Assign* powoduje skojarzenie pliku *InpFile* ze zbiorem *COMPLEX.DOC*.
• Wykonanie procedury *Reset* powoduje otwarcie pliku *InpFile*.
• Wykonanie procedury *Read* powoduje wprowadzenie z pliku *InpFile* jednej danej i przypisanie jej elementowi *ArrVar[true,3]* tablicy *ArrVar*. □

Procedura *Write*

Wywołanie: *Write(FileVar,VarList)*

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej, a *VarList* jest listą nazw zmiennych. Wymaga się, aby plik *FileVar* był otwarty. Zabrania się, aby jakakolwiek zmienna listy *VarList* była innego typu niż typ elementów pliku.

Wykonanie procedury *Write* z listą nazw zmiennych jest równoważne wykonaniu ciągu procedur *Write* z nazwami tych zmiennych. Wykonanie procedury *Write* z nazwą zmiennej powoduje wyprowadzenie do pliku *FileVar* danej przypisanej tej zmiennej.

Przykład

```
type
  ElmType = record
              Re,Im : real
            end;
var
  OutFile : file of ElmType;
  ArrVar : array[boolean] of ElmType;
...
Assign(OutFile,'COMPLEX.RES');
Rewrite(OutFile);
Write(OutFile,ArrVar[false],ArrVar[true]);
```

• Wykonanie procedury *Assign* powoduje skojarzenie pliku *OutFile* ze zbiorem *COMPLEX.RES*.
• Wykonanie procedury *Rewrite* powoduje otwarcie pliku *OutFile*.
• Wykonanie procedury *Write* powoduje wyprowadzenie do pliku *OutFile* danych przypisanych elementom *ArrVar[false]* i *ArrVar[true]* tablicy *ArrVar*.
• Zastąpienie przytoczonej instrukcji *Write* instrukcją

```
Write(OutFile,ArrVar);
```

byłoby niepoprawne, ponieważ sama nazwa tablicy nie stanowi wyszczególnienia jej elementów. □

Procedura *Seek*

Wywołanie: *Seek(FileVar,PosExp)*

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej, a *PosExp* jest wyrażeniem typu integer. Wymaga się, aby plik *FileVar* był otwarty. Zabrania się, aby wartość wyrażenia *PosExp* wykraczała poza przedział domknięty [0 ; *FileSize(FileVar)*].

Wykonanie procedury *Seek* powoduje ustawienie pliku w takiej pozycji, aby najbliższym elementem pliku był element o numerze porządkowym określonym przez wartość wyrażenia *PosExp*. Jeśli wyrażenie to ma wartość 0, to plik zostanie ustawiony w pozycji początkowej, a jeśli ma wartość *FileSize(FileVar)*, to zostanie ustawiony w pozycji końcowej.

Przykład

```
type
  Name = string[5];
var
  FileVar : file of Name;
  ElmVar1,ElmVar2 : Name;
  Len,Pos : integer;
begin
  Assign(FileVar,'FAMILY');
  Reset(FileVar);
  Len := FileSize(FileVar)-1;
  for Pos := 0 to Len shr 1 do begin
    Seek(FileVar,Pos);
    Read(FileVar,ElmVar1);
    Seek(FileVar,Len-Pos);
    Read(FileVar,ElmVar2);
    Seek(FileVar,Pos);
    Write(FileVar,ElmVar2);
    Seek(FileVar,Len-Pos);
    Write(FileVar,ElmVar1)
  end;
  Close(FileVar)
end.
```

• Wykonanie przytoczonego programu powoduje odwrócenie porządku elementów zbioru *FAMILY.*
• Otwarcie pliku *FileVar* zrealizowano za pomocą procedury Reset, ponieważ użycie procedury *Rewrite* spowodowałoby utratę zawartości zbioru *FAMILY.* □

Procedura *Close*

Wywołanie: *Close(FileVar)*

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej. Nie wymaga się, aby plik *FileVar* był otwarty.

Wykonanie procedury *Close* powoduje zamknięcie pliku *FileVar*. Jeśli przed wykonaniem tej procedury plik nie był otwarty, to jego stan nie zmienia się.

Należy nadmienić, że zakończenie wykonywania programu nie implikuje domniemanych wywołań procedury *Close.*

Przykład

```
var
  NewFile : file of boolean;
```

```
begin
  Assign(NewFile,'EMPTY');
  Rewrite(NewFile);
  Close(NewFile)
end.
```

• Wykonanie przytoczonego programu pozostawia w domniemanym katalogu, domniemanej stacji dyskowej, pusty zbiór *EMPTY*.
• W przyszłości zbiór ten może być wypełniony elementami typu *boolean*. □

Procedura *Erase*

Wywołanie: *Erase(FileVar)*

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej. Zaleca się, aby plik *FileVar* nie był otwarty.

Wykonanie procedury *Erase* powoduje usunięcie zbioru danych skojarzonego z plikiem *FileVar*.

Przykład

```
var
  KillFile : file of byte;
begin
  Assign(KillFile,'SECRET.DOC');
  Erase(KillFile)
end.
```

• Wykonanie przytoczonego programu powoduje usunięcie z domniemanego katalogu, domniemanej stacji dyskowej, zbioru *SECRET.DOC*.
• Operacja dotyczy pliku skojarzonego ze zbiorem danych, ale nie otwartego.
• Typ elementów pliku *KillFile* jest bez znaczenia. □

Procedura *Rename*

Wywołanie: *Rename(FileVar,StrExp)*

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej, a *StrExp* jest wyrażeniem łańcuchowym. Zabrania się, aby plik *FileVar* był otwarty.

Wykonanie procedury *Rename* powoduje zmianę nazwy zbioru danych skojarzonego z plikiem *FileVar* na nazwę określoną za pomocą wyrażenia łańcuchowego *StrExp*. Wymaga się, aby ciąg znaków reprezentowany przez *StrExp* był poprawną nazwą zbioru, pozbawioną określenia stacji dyskowej oraz katalogu, a ponadto, aby nie był nazwą zbioru już istniejącego. Zabrania się, aby plik *FileVar* był otwarty.

Przykład

```
var
  RenFile : file of byte;
begin
  Assign(RenFile,'SECRET.DOC');
  Rename(RenFile,'SECRET.BAK')
end.
```

• Wykonanie przytoczonego programu powoduje zmianę nazwy zbioru *SECRET.DOC* na *SECRET.BAK*.
• Typ elementów pliku *Ren File* jest bez znaczenia. □

Funkcja *Eof*

Wywołanie: *Eof (FileVar)*

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej. Wymaga się, aby plik *FileVar* był otwarty.

Rezultatem funkcji *Eof* jest dana o wartości *true* — jeśli plik *FileVar* znajduje się w pozycji końcowej, albo dana o wartości *false* — w przeciwnym razie.

Przykład

```
var
  FileVar : file of byte;
begin
  Assign(FileVar,'USELESS.DOC');
  Rewrite(FileVar);
  Writeln(Eof (FileVar))
end.
```

• Bezpośrednio po wykonaniu procedury *Rewrite* plik znajduje się zarówno w pozycji początkowej, jak i końcowej.
• Wykonanie instrukcji

Writeln(Eof (FileVar))

powoduje wyprowadzenie napisu *TRUE*. □

Funkcja *FileSize*

Wywołanie: *FileSize(FileVar)*

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej. Wymaga się, aby plik *FileVar* był otwarty.

Rezultatem funkcji *FileSize* jest dana typu *integer*. Wartością tej danej jest liczba elementów pliku *FileVar*.

Przykład

```
var
  FileVar : file of 2..4;
begin
  Assign(FileVar,'USELESS.DOC');
  Rewrite(FileVar);
  Writeln(FileSize(FileVar));
end.
```

- Bezpośrednio po wykonaniu procedury *Rewrite* plik *FileVar* jest pusty.
- Ponieważ plik pusty zawiera 0 elementów, wykonanie przytoczonego programu powoduje wyprowadzenie liczby 0. □

Funkcja *FilePos*

Wywołanie: *FilePos(FileVar)*

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej. Wymaga się, aby plik *FileVar* był otwarty.

Rezultatem funkcji *FilePos* jest dana typu *integer*. Wartością tej danej jest numer pozycji pliku *FileVar*. Numerem pozycji początkowej jest 0, a numerem pozycji końcowej jest *FileSize(FileVar)*.

Przykład

```
var
  FileVar : file of (Red,Green,Blue);
begin
  Assign(FileVar,'COLORS');
  Reset(FileVar);
  Seek(FileSize(FileVar));
  Writeln(FilePos(FileVar) = 0)
end.
```

- Jeśli zbiór *COLORS* jest pusty, to wykonanie przytoczonego programu spowoduje wyprowadzenie napisu *TRUE*.
- W przeciwnym razie nastąpi wyprowadzenie napisu *FALSE*. □

Pliki tekstowe

W odróżnieniu od pozostałych plików, pliki tekstowe nie składają się z sekwencji identycznych elementów tego samego typu, lecz składają się z wierszy podzielonych na znaki. Każdy wiersz pliku tekstowego jest zakończony parą

znaków CR/LF (carriage-return/line-feed). Bezpośrednio po ostatnim wierszu pliku występuje znak Ctrl-Z.

Ponieważ wiersze pliku mogą być różnej długości, przetwarzanie pliku może być jedynie sekwencyjne, plik zaś może zostać otwarty tylko do wyprowadzania (*Rewrite*) albo tylko do wprowadzania (*Reset*). W systemie DOS istnieje ponadto możliwość otwarcia pliku tekstowego w trybie do rozszerzania (*Append*). W takim przypadku, bezpośrednio po otwarciu, plik zostaje ustawiony w pozycji końcowej i jest traktowany tak, jakby był otwarty do wyprowadzenia.

Opis typu plikowego tekstowego składa się z predefiniowanego identyfikatora *text*.

Składnia

  opis-typu-plikowego-tekstowego:
    text

Przykłady

```
type
  TextType = text;
var
  OutFile : TextType;
  InpFile : text;
```

• Typ *TextType* jest związany ze zbiorem sekwencji wierszy podzielonych na znaki i zakończonych znakami CR/LF.
• Nazwy *OutFile* i *InpFile* reprezentują pliki tekstowe. □

W języku Turbo Pascal komunikowanie się takimi urządzeniami zewnętrznymi jak konsole, terminale, drukarki i modemy odbywa się za pomocą plików tekstowych. Pliki te stanowią zatem modele fizycznych zbiorów danych dostępnych za pomocą tych urządzeń.

Urządzenia zewnętrzne wymienionych tu typów mają swoje symboliczne oznaczenia, składające się z trzyliterowej mnemoniki i znaku : (dwukropek).

CON: — konsola

Konsola jest urządzeniem wejściowo-wyjściowym. Elementem wyjściowym konsoli jest ekran monitora, a elementem wejściowym — klawiatura. Wprowadzanie danych z konsoli jest *buforowane*. Oznacza to, że dane są wprowadzane z konsoli pełnymi wierszami i dopiero po wprowadzeniu całego wiersza, stanowiące go znaki podlegają przetwarzaniu. Ponieważ każdy wiersz jest kończony znakiem CR, możliwe jest wprowadzenie z konsoli ciągu

znaków, a następnie — ale przed wprowadzeniem znaku CR — poddanie go edycji. Edycja może być realizowana za pomocą następujących znaków:

Ctrl-H:
: Usunięcie znaku znajdującego się z lewej strony kursora i przesunięcie kursora o jedną pozycję w lewo.

Ctrl-X:
: Usunięcie wszystkich znaków znajdujących się z lewej strony kursora i przesunięcie kursora do pierwszej kolumny wiersza.

Ctrl-D:
: Wstawienie na pozycję kursora jednego znaku z poprzedniego wiersza i przesunięcie kursora o jedną pozycję w prawo.

Ctrl-R:
: Wstawienie, począwszy od pozycji kursora, pozostałych znaków poprzedniego wiersza i przesunięcie kursora w prawo za ostatni wstawiony znak.

CR, Ctrl-M:
: Zakończenie wprowadzania wiersza i umieszczenie w buforze wejściowym pary znaków CR/LF.

Ctrl-Z:
: Zakończenie wprowadzania wiersza i umieszczenie w buforze wejściowym znaku Ctrl-Z.

Należy nadmienić, że rozmiar bufora wejściowego konsoli jest określony przez predefiniowaną zmienną *BufLen*. Zarówno maksymalną jak i domniemaną wartością tej zmiennej jest 127. Ustalenie nowej wartości zmiennej *BufLen* jest skuteczne jedynie w odniesieniu do najbliższej instrukcji wprowadzania. Bezpośrednio po wykonaniu każdej takiej instrukcji zmienna *BufLen* otrzymuje ponownie wartość 127.

TRM: — terminal

Terminal jest urządzeniem wejściowo-wyjściowym. Elementem wyjściowym terminala jest ekran monitora, a elementem wejściowym — klawiatura. W odróżnieniu od konsoli, wprowadzanie danych z terminala nie jest buforowane. Oznacza to, że każdy wprowadzony znak natychmiast podlega przetwarzaniu. Jednocześnie ze skierowaniem do przetwarzania znak jest wyprowadzany na ekran. Spośród znaków sterujących dotyczy to jednak tylko znaku CR, który jest wyprowadzany jako para CR/LF.

KBD: — klawiatura

Klawiatura jest urządzeniem wejściowym. Znaki wprowadzane z urządzenia KBD: pochodzą z elementu wejściowego konsoli, nie podlegają buforowaniu i nie są wyprowadzane na ekran.

LST: — drukarka

Drukarka jest urządzeniem wyjściowym. Wyprowadzane znaki nie podlegają buforowaniu w systemie, ale mogą być buforowane w drukarce.

AUX: — urządzenie pomocnicze

Urządzenie pomocnicze jest urządzeniem wejściowym albo wyjściowym. W systemie DOS pełni ono funkcję urządzenia COM1:.

USR: — urządzenie użytkownika

Urządzenie użytkownika jest urządzeniem programowanym. Umożliwia ono ingerencję w proces przesyłania znaków. Jego użycie wiąże się z projektowaniem własnych programów obsługi transmisji znaków.

Skojarzenie pliku z urządzeniem logicznym może być dokonane za pomocą procedury *Assign*, w której wywołaniu jest podawana nazwa zmiennej plikowej oraz wyrażenie łańcuchowe określające nazwę urządzenia logicznego.

W odróżnieniu od skojarzenia pliku ze zbiorem skojarzenie pliku z urządzeniem logicznym pociąga za sobą niejawne otwarcie pliku. Z tego względu posłużenie się procedurami *Reset* i *Rewrite* jest zbyteczne, a ich wykonanie, podobnie jak wykonanie procedury *Close*, nie ma żadnych skutków. Błędne jest natomiast posłużenie się takimi procedurami jak *Erase* i *Rename*, gdyż można je stosować jedynie w odniesieniu do zbiorów danych w pamięci dyskowej.

Przykład

```
var
  Console : text;
...
Assign(Console,'CON:');
```

• Wykonanie procedury *Assign* spowoduje skojarzenie pliku tekstowego *Console* z urządzeniem logicznym CON:, tj. konsolą, a następnie niejawne otwarcie tego pliku.

• Ponieważ wykonanie procedury *Close* dotyczącej pliku skojarzonego z urządzeniem logicznym nie ma żadnych skutków, zamknięcie rozpatrywanego pliku tekstowego byłoby możliwe dopiero po wykonaniu innej procedury *Assign* dotyczącej zmiennej *Console*, np.

```
Assign(Console,'KBD:');
```
□

W celu ułatwienia posługiwania się plikami skojarzonymi z urządzeniami logicznymi wprowadzono w Turbo Pascalu szereg predefiniowanych zmiennych plikowych, reprezentujących pliki tekstowe i dokonano niejawnych skojarzeń tych plików z wybranymi urządzeniami logicznymi.

Jak wynika z tabl. 12.1 z predefiniowanymi nazwami zmiennych plikowych są związane ustalone urządzenia logiczne. Wyjątek od tej zasady dotyczy zmiennych plikowych *Input* i *Output*, z których każda może reprezentować plik skojarzony z urządzeniem CON: albo z urządzeniem TRM:.

Tablica 12.1. Skojarzenia plików tekstowych i urządzeń logicznych

Zmienna plikowa	Urządzenie logiczne
Input	CON: albo TRM:
Output	CON: albo TRM:
Con	CON:
Trm	TRM:
Kbd	KBD:
Lst	LST:
Aux	AUX:
Usr	USR:

Wybór CON: albo TRM: odbywa się na podstawie dyrektywy kompilatora {$*B*+} albo {$*B*−}. Przez domniemanie przyjmuje się {$*B*+}, kiedy to plik reprezentowany przez *Input* jak i przez *Output* jest kojarzony z urządzeniem CON:. W zasięgu dyrektywy {$*B*−} odbywa się natomiast skojarzenie z urządzeniem TRM:.

Należy zwrócić uwagę, że pliki reprezentowane przez nazwy wymienionych zmiennych plikowych są zawsze otwarte, a operacje na nich dotyczą ustalonych urządzeń logicznych.

Przykład

```
begin
  Writeln(Con,'Hello world')
end
```

- *Con* jest predefiniowaną zmienną plikową, reprezentującą plik skojarzony z konsolą.
- Wykonanie przytoczonego programu powoduje wyprowadzenie na konsolę napisu *Hello world.* □

Operacje na plikach tekstowych

Jak już wyjaśniono, pliki tekstowe składają się z wierszy podzielonych na znaki. Każdy wiersz jest zakończony parą znaków CR/LF, a ostatnim znakiem pliku jest Ctrl-Z. Znak Ctrl-Z jest umieszczany w pliku otwartym do wyprowadzania lub rozszerzania — w chwili zamykania pliku.

Pliki tekstowe mogą być kojarzone ze zbiorami danych albo z urządzeniami logicznymi. W pierwszym przypadku przetwarzanie danych zawartych w pliku musi być poprzedzone wykonaniem procedury *Assign* i jednej z procedur *Reset*, *Rewrite*, *Append*. Na zakończenie tego przetwarzania powinna być wykonana procedura *Close*. W drugim przypadku, tj. skojarzenia pliku z urządzeniem, można posłużyć się predefiniowaną zmienną plikową. W takim przypadku zabrania się wykonania procedur *Assign*, *Reset*, *Rewrite*, *Close* dotyczących tej zmiennej, albo też można potraktować urządzenie tak jakby było zbiorem danych o nazwie symbolicznej urządzenia, posłużyć się procedurą *Assign* dla skojarzenia pliku z urządzeniem, a następnie przystąpić do przetwarzania pliku.

Procedura *Assign*

Wywołanie: *Assign*(*TextVar*,*StrExp*)

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*, a *StrExp* jest wyrażeniem łańcuchowym. Wymaga się, aby plik reprezentowany przez *TextVar* nie był otwarty. Zabrania się, aby *TextVar* było nazwą predefiniowanej zmiennej plikowej.

Wykonanie procedury *Assign* powoduje skojarzenie pliku *TextVar* ze zbiorem danych albo z urządzeniem logicznym o nazwie określonej przez *StrExp*.

Przykład

```
var
  Device : text;
...
Assign(Device,'CON:');
```

- Nazwa zmiennej plikowej *Device* reprezentuje plik tekstowy.
- Wykonanie procedury *Assign* powoduje skojarzenie pliku *Device* z konsolą.
- Wykonanie procedury *Assign* powoduje otwarcie pliku *Device*. □

Procedura *Reset*

Wywołanie: *Reset*(*TextVar*)

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*. Wymaga się, aby przed wywołaniem procedury *Reset* plik *TextVar* był skojarzony z istniejącym zbiorem danych albo z urządzeniem logicznym. Zabrania się, aby *TextVar* było nazwą predefiniowanej zmiennej plikowej.

Wykonanie procedury *Reset* powoduje otwarcie pliku *TextVar*. Jeśli plik *TextVar* jest skojarzony z urządzeniem logicznym, to jest już otwarty i użycie procedury *Reset* nie ma żadnych skutków.

Przykład

```
var
  InpFile : text;
...
Assign(InpFile,'OLDBOOK');
Reset(InpFile);
```

• Wykonanie procedury *Assign* powoduje skojarzenie pliku tekstowego *InpFile* ze zbiorem *OLDBOOK*.
• Wykonanie procedury *Reset* powoduje otwarcie pliku tekstowego *InpFile* w trybie do wprowadzania.

Procedura *Rewrite*

Wywołanie: *Rewrite*(*TextVar*)

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*. Wymaga się, aby przed wywołaniem procedury *Rewrite* plik *TextVar* był skojarzony ze zbiorem danych albo z urządzeniem logicznym. Zabrania się, aby *TextVar* było nazwą predefiniowanej zmiennej plikowej.

Wykonanie procedury *Rewrite* powoduje otwarcie pliku *TextVar* w trybie do wprowadzania. Jeśli plik *TextVar* jest skojarzony z urządzeniem logicznym, to jest już otwarty i użycie procedury *Rewrite* nie ma żadnych skutków. Jeśli plik *TextVar* jest skojarzony ze zbiorem danych, a zbiór ten nie istnieje, to zostanie utworzony. Jeśli istnieje, to zostanie usunięty i utworzony ponownie. W obu przypadkach zostanie utworzony zbiór pusty.

Przykład

```
var
  OutFile : text;
...
Assign(OutFile,'NEWBOOK');
Rewrite(OutFile);
```

• Wykonanie procedury *Assign* powoduje skojarzenie pliku tekstowego *OutFile* ze zbiorem *NEWBOOK*.
• Wykonanie procedury *Rewrite* powoduje otwarcie pliku *OutFile* w trybie do wyprowadzania. Po dokonaniu otwarcia rozpatrywany plik jest pusty. □

Procedura *Append*

Wywołanie: *Append*(*TextVar*)

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*. Wymaga się, aby przed wywołaniem procedury *Append* plik *TextVar* był skojarzony z ist-

niejącym już zbiorem danych albo z urządzeniem logicznym. Zabrania się, aby *TextVar* było nazwą predefiniowanej zmiennej plikowej.

Wykonanie procedury *Append* powoduje otwarcie pliku *TextVar*. Jeśli plik reprezentowany przez *TextVar* jest skojarzony z urządzeniem logicznym, to jest już otwarty i użycie procedury *Append* nie ma żadnych skutków. Jeśli plik *TextVar* jest skojarzony ze zbiorem danych, to zostanie otwarty i ustawiony w pozycji końcowej.

Przykład

```
var
  Extend : text;
...
Assign(Extend,'BOOK');
Append(Extend);
```

- Wykonanie procedury *Assign* powoduje skojarzenie pliku tekstowego *Extend* ze zbiorem *BOOK*.
- Wykonanie procedury *Append* powoduje otwarcie pliku *Extend* w trybie do rozszerzania. □

Procedura *Read*

Wywołanie: *Read*(*TextVar*,*VarList*)

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*, a *VarList* jest nazwą zmiennej albo listą nazw zmiennych typu *char*, **string**, *integer* lub *real*. Jeśli *TextVar* jest nazwą *Input*, to wywołanie może zostać uproszczone do *Read*(*VarList*). Jeśli wywołanie z jawną albo domniemaną nazwą *Input* znajduje się w zasięgu dyrektywy {$*B*+}, to jest traktowane tak, jakby *TextVar* było nazwą *Con*. Jeśli znajduje się w zasięgu dyrektywy {$*B*−}, to jest traktowane tak, jakby *TextVar* było nazwą *Trm*. W każdym przypadku wymaga się, aby plik reprezentowany przez zmienną *TextVar* był otwarty.

Wykonanie procedury *Read* powoduje wprowadzenie ciągu znaków z pliku *TextVar*, zinterpretowanie ich jako umownych zapisów wartości danych, a następnie przypisanie danych o tych wartościach zmiennym reprezentowanym przez *VarList*.

Należy nadmienić, że jeśli wprowadzanie dotyczy urządzenia logicznego CON:, to każde wykonanie procedury *Read* powoduje wprowadzenie nowego wiersza tekstu, i to nawet wtedy, gdy w buforze konsoli pozostają nie wprowadzone jeszcze znaki. W takim przypadku, interpretowanie znaków bufora następuje dopiero po zakończeniu wiersza znakiem CR, a ewentualne

wcześniejsze wprowadzenie znaku Ctrl-Z jest ignorowane. Znak taki jest zawsze umieszczany na końcu bufora, ale dopiero po wprowadzeniu znaku CR.

Sposób interpretowania znaków pliku *TextVar* zależy od typu zmiennych *VarList*.

Dla zmiennych typu *char*:
: Wprowadzany jest jeden znak i przypisywany zmiennej.

Dla zmiennych typu **string**:
: Wprowadzany jest najdłuższy dopuszczalny ciąg znaków i przypisywany zmiennej. Liczba wprowadzonych znaków nie przekracza maksymalnego rozmiaru zmiennej łańcuchowej, ani nie jest większa niż liczba znaków bieżącego wiersza.

Dla zmiennych typu *integer*:
: Wprowadzany jest ciąg znaków mający postać liczby całkowitej, po której następuje jeden ze znaków: spacja, tabulacja, CR albo Ctrl-Z. Znaki spacji, tabulacji, CR i LF poprzedzające wspomnianą liczbę są ignorowane. Następnie dana o wartości wprowadzonej liczby zostaje przypisana kolejnej zmiennej *VarList*. Jeśli wprowadzony ciąg znaków ma inną postać niż wynika to z opisu, powstaje błąd wykonania operacji wejścia/wyjścia.

Dla zmiennych typu *real*:
: Wprowadzany jest ciąg znaków mający postać liczby rzeczywistej, po której następuje jeden ze znaków: spacja, tabulacja, CR albo Ctrl-Z. Znaki spacji, tabulacji, CR i LF poprzedzające liczbę są ignorowane. Następnie dana o wartości wprowadzonej liczby zostaje przypisana kolejnej zmiennej *VarList*. Jeśli wprowadzony ciąg znaków ma inną postać niż wynika to z opisu, powstaje błąd wykonania operacji wejścia/wyjścia.

Przykład

```
var
  IntVar : integer;
  FixVar : real;
  TextVar : text;
  Ter1,Ter2 : char;
  Str : string[8];
begin
  Assign(TextVar,'ONELINE.DOC');
  Reset(TextVar);
  Read(TextVar,IntVar,Ter1,FixVar,Ter2,Str);
  ...
end
```

• Jeśli przyjąć, że pierwszym wierszem tekstu zawartym w zbiorze *ONELINE.DOC* jest ciąg

12 — 13.8 *Izabela*

to zmiennej *IntVar* zostanie przypisana dana o wartości 12, zmiennej *FixVar* dana o wartości —13.8, a zmiennej Str dana o wartości '*Izabela*'.

• Każdej ze zmiennych *Ter1* i *Ter2* zostanie przypisana spacja. Oznacza to, że znak kończący liczbę podlega podwójnej interpretacji, jako ostatni znak zapisu liczby i jako pierwszy znak ciągu następującego po liczbie. □

Jeśli podczas wykonywania procedury *Read* plik znajdzie się w pozycji końcowej, a nie wszystkim zmiennym *VarList* zostały już przypisane dane, to każdej zmiennej typu *char* zostanie przypisany znak Ctrl-Z, każdej zmiennej typu **string** zostanie przypisany pusty ciąg znaków, a zmiennym typu *integer* i *real* nie zostaną przypisane żadne dane.

Przykład

```
var
  IntVar : integer;
  FixVar : real;
  TextVar : text;
  Ter1,Ter2 : char;
  Str : string[4];
begin
  IntVar:= -44;
  FixVar:= -44;
  Str:= 'janb';
  Ter1 := 'x';
  Ter2:= 'y';
  Assign(TextVar,'SHORT.DOC');
  Reset(TextVar);
  Read(TextVar,IntVar,Ter1,FixVar,Ter2,Str);
  ...
end.
```

• Jeśli przyjąć, że zbiór *SHORT.DOC* składa się ze znaków: 1, 3, spacja, CR, LF, Ctrl-Z, to po wykonaniu instrukcji *Read* zmienna *IntVar* ma wartość 13, zmienna *FixVar* ma wartość —44.0, zmienna *Ter1* ma wartość #32 (spacja), zmienna *Ter2* ma wartość #26, (Ctrl-Z), a zmienna *Str* ma wartość ''.

Należy nadmienić, że jeśli plik jest skojarzony z konsolą, to na końcu każdego wprowadzonego wiersza jest umieszczany znak Ctrl-Z. Ma to taki skutek, jakby przed końcem każdego wiersza plik znajdował się w pozycji końcowej.

Przykład

```
var
  IntVar : integer;
  StrVar : string[3];
  ChrVar : char;
begin
  IntVar:= -2;
  StrVar:= 'jan';
  ChrVar:= 'b':
  Read(IntVar,StrVar,ChrVar);
  ...
end.
```

- Wywołanie procedury *Read* jest interpretowane tak jak wywołanie

  Read(*Input*,*IntVar*,*StrVar*,*ChrVar*);

- Ponieważ wymienione wywołanie znajduje się w zasięgu domniemanej dyrektywy {$*B*+}, jest interpretowane tak jak wywołanie

  Read(*Con*,*IntVar*,*StrVar*,*ChrVar*);

- Jeśli podczas wykonywania procedury *Read* zostanie wprowadzony z konsoli tylko jeden znak CR, to spowoduje to zakończenie wykonywania tej instrukcji. W tym momencie zmienna *IntVar* będzie miała wartość −2, zmienna *StrVar* będzie miała wartość '', a zmienna *ChrVar* będzie miała wartość #26 (Ctrl-Z).

Procedura *Readln*

Wywołanie: *Readln*(*TextVar*)
Readln(*TextVar*, *VarList*)

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*, a *VarList* jest listą nazw zmiennych typu *char*, **string**, *integer* lub *real*. Jeśli *TextVar* jest nazwą *Input*, to wywołanie w pierwszej postaci może być uproszczone do *Readln*, a w drugiej do *Readln*(*VarList*).

Wywołanie procedury *Readln* w postaci *Readln*(*TextVar*) powoduje wprowadzenie i zignorowanie ciągu znaków pliku aż do CR/LF włącznie. Jeśli plik jest skojarzony z urządzeniem logicznym, to są pomijane znaki tylko do CR włącznie. Jeśli *TextVar* reprezentuje konsolę, to na ekran monitora zostaną wyprowadzone znaki CR/LF. Natomiast wywołanie tej procedury w postaci

Readln(*TextVar*, *VarList*)

jest równoważne parze wywołań

Read(TextVar,VarList); Readln(TextVar)

i z tego powodu nie wymaga dodatkowych wyjaśnień.

Przykład

```
var
  Count : integer;
  Source : text;
  Line : string[128];
begin
  Assign(Source,'VOLUME.DOC');
  Reset(Source);
  Count:= 0;
  while not Eof(Source) do begin
    Readln(Source,Line);
    Count:= Count+1
  end;
  Writeln(Count)
end
```

• Wykonanie przytoczonego programu powoduje wyznaczenie liczby wierszy zbioru danych *VOLUME.DOC*. □

Procedura *Write*

Wywołanie: *Write(TextVar,ExpList)*

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*, a *ExpList* jest listą wyrażeń typu *char*, **string**, *integer*, *real* albo *boolean*. Bezpośrednio po każdym z wymienionych wyrażeń może wystąpić *kwalifikator* o postaci :*m*, a po wyrażeniu typu *real* także kwalifikator o postaci :*m*:*n*, w którym *m* i *n* są wyrażeniami typu *integer*. Jeśli *TextVar* jest nazwą *Output*, to wywołanie może zostać uproszczone do *Write(ExpList)*. Wywołanie z jawną albo domniemaną nazwą *Output* jest traktowane tak, jakby *TextVar* było nazwą *Con* (lub równoważną jej w danym kontekście nazwą *Trm*). W każdym przypadku wymaga się, aby plik *TextVar* był otwarty.

Wykonanie procedury *Write* powoduje wprowadzenie do pliku *TextVar*, ciągu znaków składającego się z umownych zapisów wartości danych reprezentowanych przez wyrażenia listy *ExpList*.

Sposób tworzenia znaków wspomnianego ciągu zależy od typu wyrażeń *ElmList* oraz od wartości kwalifikatorów *m* i *n*.

Dla wyrażeń typu *char*:

Wyprowadzany jest znak reprezentowany przez wyrażenie. Jeśli posłużono się kwalifikatorem *m*, a $m > 1$, to znak ten zostanie poprzedzony $m - 1$ spacjami.

Dla wyrażeń typu **string**:

Wyprowadzany jest ciąg znaków reprezentowany przez wyrażenie. Jeśli posłużono się kwalifikatorem *m*, a *m* jest większe niż liczba znaków *l* tego ciągu, to ciąg ten zostanie poprzedzony $m - l$ spacjami.

Dla wyrażeń typu *integer*:

Wyprowadzany jest ciąg znaków o postaci najkrótszego literału reprezentującego wartość wyrażenia. Jeśli posłużono się kwalifikatorem *m*, a *m* jest większe niż liczba znaków *l* wspomnianego literału, to ciąg ten zostanie poprzedzony $m - l$ spacjami.

Dla wyrażeń typu *real*:

Wyprowadzany jest ciąg znaków o postaci *bsd.dddddddddEsdd* reprezentujący wartość wyrażenia. W ciągu tym *b* jest spacją, a pierwsze *s* jest znakiem przedstawianym jako spacja dla wyrażenia o wartości nieujemnej, a jako znak – (minus) dla wyrażenia o wartości ujemnej. Każde *d* reprezentuje w tym ciągu cyfrę dziesiętną, a drugie *s* jest znakiem + (plus) albo – (minus). Jeśli posłużono się kwalifikatorem *m*, a $m > 18$, to przedstawiony ciąg znaków zostanie dodatkowo poprzedzony $m - 18$ spacjami. Jeśli $m = 18$, to zostanie wyprowadzony ciąg wzorcowy podany na początku opisu, jeśli natomiast $m < 18$, to wyprowadzony ciąg zostanie skrócony do *m* znaków, pochodzących z ciągu wzorcowego przez odrzucenie z niego w pierwszej kolejności spacji wiodących, a następnie końcowych cyfr mantysy, ale z zaokrągleniem. Jeśli wartość *m* jest taka, że wymagałoby to pozostawienia mniej niż 2 cyfr mantysy, poprzestaje się na odrzuceniu cyfry trzeciej i następnych, także z zaokrągleniem. Jeśli posłużono się kwalifikatorem o postaci $:m:n$, to jest wyprowadzany ciąg znaków mający postać literału rzeczywistego bez wykładnika, z *n* cyframi ułamkowymi, wyrównanego prawostronnie w ciągu o długości *m* znaków. Jeśli wspomniany literał nie da się przedstawić za pomocą ciągu o tej liczbie znaków, to *m* zostanie niejawnie zwiększone o tyle, aby było to wykonalne. Należy przy tym uwzględnić, że jeśli $n = 0$, to literał jest wyprowadzany również bez części ułamkowej i kropki, a jeśli *n* nie należy do przedziału domkniętego [0 ; 24], to kwalifikator $:m:n$ jest traktowany tak jak uprzednio opisany kwalifikator $:m$.

Dla wyrażeń typu *boolean*:

Wyprowadzany jest ciąg znaków o postaci *TRUE* albo *FALSE*, reprezentujący wartość wyrażenia. Jeśli posłużono się kwalifikatorem *m*, a *m* jest większe niż liczba *l* znaków tego ciągu, to ciąg zostanie poprzedzony $m - l$ spacjami.

Przykłady

(Symbol *b* oznacza spację.)

Wyrażenie	Ciąg wyprowadzany
'j'	*j*
'j' : 2	*bj*
'jan'	*jan*
'jan' : 4	*bjan*
44	44
−44	−44
44 : 4	*bb*44
23.5	*bb*2.3500000000*E*+01
23.456789 :10	2.3457*E*+01
23.456789 :6	2.3*E*+01
23.456789 :6:2	*b*23.46
−23.456789 :6:0	*bbb*−23
23.456789 :6:−2	2.3*E*+01

● Jak wynika z przykładów, zaokrąglenie jest dokonywane na podstawie najbardziej znaczącej cyfry odrzucanej. □

Procedura *Writeln*

Wywołanie: *Writeln(TextVar)*
Writeln(TextVar,ExpList)

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*, a *ExpList* jest listą nazw zmiennych typu *char*, **string**, *integer*, *real* lub *boolean*. Bezpośrednio po każdym z wymienionych wyrażeń może wystąpić kwalifikator o postaci :*m*, a po wyrażeniu typu *real* także kwalifikator o postaci :*m*:*n*, gdzie *m* i *n* są wyrażeniami typu *integer*. Jeśli *TextVar* jest nazwą *Output*, to wywołanie w pierwszej postaci może być uproszczone do *Writeln*, a w drugiej do *Writeln(ExpList)*.

Wywołanie procedury *Writeln* w postaci *Writeln(TextVar)* powoduje wyprowadzenie do pliku *TextVar* pary znaków CR/LF. Natomiast wywołanie tej procedury w postaci

Writeln(TextVar,ExpList)

jest równoważne parze wywołań

Write(TextVar,ExpList); Writeln(TextVar);

i z tego powodu nie wymaga dodatkowych wyjaśnień.

Przykład

```
begin
  Writeln(false);
  Writeln;
  Writeln(true)
end.
```

• Wykonanie przytoczonego programu powoduje wyprowadzenie na konsolę 3 wierszy tekstu.
• Drugi z wyprowadzonych wierszy jest pusty.
• Ogółem zostanie wyprowadzonych 15 znaków, w tym napisy *FALSE* i *TRUE* oraz dwie pary znaków CR/LF. □

Procedura *Close*

Wywołanie: *Close*(*TextVar*)

Przyjmuje się, że *TextVar* jest nazwą znmiennej plikowej typu *text*. Zabrania się, aby *TextVar* było nazwą predefiniowanej zmiennej plikowej.

Wywołanie procedury *Close* powoduje zamknięcie pliku *TextVar*.

Przykład

```
var
  Output : text;
begin
  Assign(Output,'JB.DOC');
  Rewrite(Output);
  Writeln(Output,'jasio');
  Close(Output)
end.
```

• Wobec jawnej deklaracji nazwa *Output* nie jest nazwą predefiniowanej zmiennej plikowej, więc wywołanie procedury *Assign* i *Rewrite* jest poprawne i niezbędne.
• Wykonanie przytoczonego programu powoduje utworzenie zbioru zawierającego 8 znaków.
• Wywołanie procedury *Close* powoduje wyprowadzenie znaku Ctrl-Z oraz wykonanie czynności kończących utworzenie zbioru *JB.DOC* zawierającego ciąg znaków: jasio CR LF Ctrl-Z. □

Funkcja *Eof*

Wywołanie: *Eof*(*TextVar*)

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*. Jeśli *TextVar* jest nazwą *Input*, to wywołanie może być uproszczone do *Eof*.

Wywołanie funkcji *Eof* może dotyczyć zarówno pliku skojarzonego ze zbiorem danych, jak i pliku skojarzonego z urządzeniem logicznym. W obu przypadkach rezultatem funkcji jest dana typu *boolean* o wartości *true* albo *false*.

Jeśli plik jest skojarzony ze zbiorem danych, to rezultatem funkcji *Eof* jest dana o wartości *true* — jeśli plik znajduje się w pozycji przed znakiem Ctrl-Z albo w pozycji końcowej. W przeciwnym razie rezultatem tej funkcji jest dana o wartości *false*.

Jeśli plik jest skojarzony z urządzeniem logicznym, to rezultatem funkcji *Eof* jest dana o wartości *true* — jeśli ostatnim zinterpretowanym znakiem był Ctrl-Z. W przeciwnym razie rezultatem tej funkcji jest dana o wartości *false*.

Przykład

```
var
  ChrVar : char;
begin
  Read(ChrVar);
  Writeln(Eof)
end.
```

• Jeśli podczas wykonywania przytoczonego programu zostaną wprowadzone z konsoli znaki Ctrl-Z i CR, to nastąpi wyprowadzenie napisu *FALSE*.
• Stanie się tak dlatego, że znak Ctrl-Z zostanie pominięty, a więc rezultatem funkcji *Eof* będzie dana o wartości *false*. □

Funkcja *Eoln*

Wywołanie: *Eoln*(*TextVar*)
Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*. Jeśli *TextVar* jest nazwą *Input*, to wywołanie może zostać uproszczone do *Eoln*.

Wywołanie funkcji *Eoln* może dotyczyć zarówno pliku skojarzonego ze zbiorem danych, jak i pliku skojarzonego z urządzeniem logicznym. W obu przypadkach rezultatem funkcji jest dana typu *boolean* o wartości *true* albo *false*.

Jeśli plik jest skojarzony ze zbiorem danych, to rezultatem funkcji *Eoln* jest dana o wartości *true* — jeśli plik znajduje się w pozycji przed znakiem CR, albo gdy rezultatem funkcji *Eof* jest dana o wartości *true*. W przeciwnym razie rezultatem funkcji *Eoln* jest dana o wartości *false*.

Jeśli plik jest skojarzony z urządzeniem logicznym, to rezultatem funkcji *Eoln*

jest dana o wartości *true* — jeśli ostatnim zinterpretowanym znakiem był CR, albo gdy rezultatem funkcji *Eof* jest dana o wartości *true*. W przeciwnym razie rezultatem funkcji *Eoln* jest dana o wartości *false*.

Przykład

```
var
  ChrVar : char;
  TxtVar : text;
begin
  Assign(TxtVar,'STRANGE.DOC');
  Reset(TxtVar);
  while not Eof(TxtVar) do begin
    Writeln(Eoln(TxtVar));
    Read(TxtVar,ChrVar)
  end
end.
```

• Jeśli zbiór *STRANGE.DOC* składa się z jednego wiersza pustego, tj. ze znaków CR, LF, Ctrl-Z, to wykonanie przytoczonego programu powoduje wyprowadzenie napisu

TRUE
FALSE

• Wynika stąd, że w pozycji przed znakiem LF rezultatem funkcji *Eoln* jest dana o wartości *false*.

Funkcja *SeekEof*

Wywołanie: *SeekEof*(*TextVar*)

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*. Jeśli *TextVar* jest nazwą *Input*, to wywołanie może zostać uproszczone do *SeekEof*.

Wywołanie funkcji *SeekEof* jest zbliżone do wywołania funkcji *Eof*. Bezpośrednio po pominięciu najbliższych spacji, tabulacji oraz znaków CR i LF jest udostępniany taki sam rezultat jak dla funkcji *Eof*.

Przykład

```
var
  TxtVar : text;
begin
  Assign(TxtVar,'STRANGE.DOC');
  Reset(TxtVar);
  Writeln(SeekEof(TxtVar))
end.
```

• Jeśli zbiór *STRANGE.DOC* składa się z jednego wiersza pustego to wykonanie przytoczonego programu powoduje wyprowadzenie napisu *TRUE*. □

Funkcja *SeekEoln*

Wywołanie: *SeekEoln(TextVar)*

Przyjmuje się, że *TextVar* jest nazwą zmiennej plikowej typu *text*. Jeśli TextVar jest nazwą *Input*, to wywołanie może zostać uproszczone do *SeekEoln*.

Wykonanie funkcji *SeekEoln* jest zbliżone do wykonania funkcji *Eoln*. Bezpośrednio po minięciu najbliższych spacji i tabulacji jest udostępniany taki sam rezultat jak dla funkcji *Eoln*.

Przykład

```
var
  TxtVar : text;
  Tally : integer;
  ChrVar : char;
begin
  Tally:= 0;
  Assign(TxtVar,'TEXT.DOC');
  Reset(TxtVar);
  while not SeekEof(TxtVar) do begin
    while not SeekEoln(TxtVar) do begin
      Tally:= Tally+1;
      Read(TxtVar,ChrVar)
    end
  end;
  Writeln(Tally)
end.
```

• Wykonanie przytoczonego programu powoduje wyznaczenie liczby znaków zbioru *TEXT.DOC*.
• Spacje, tabulacje, znaki CR, LF i Ctrl-Z kończące wiersze są ignorowane. □

Pliki blokowe

Pliki blokowe umożliwiają wykonywanie niebuforowanych operacji wprowadzania/wyprowadzania, dokonywanych bezpośrednio między zmiennymi programu a zewnętrzną pamięcią dyskową. Przyjmuje się, że elementami pliku blokowego są bloki po 128 bajtów. Plik blokowy może reprezentować dowolny

zbiór dyskowy. Z tego względu operacje takie jak np. *Erase* lub *Rename* mogą być wykonywane za pośrednictwem plików blokowych.

Opis typu pliku blokowego składa się ze słowa kluczowego **file**.

Składnia

opis-typu-plikowego-blokowego:
file

Przykład

```
var
  DiskFile : file;
  FileName : string[40];
begin
  Write('FileName :— ');
  Readln(FileName);
  Assign(DiskFile,FileName);
  Erase(DiskFile)
end.
```

• Wykonanie przytoczonego programu powoduje usunięcie dowolnego zbioru dyskowego o nazwie wprowadzonej z klawiatury konsoli.
• Typ elementów zbioru jest bez znaczenia. □

Wykonywanie operacji wprowadzania/wyprowadzania na plikach blokowych jest realizowane za pomocą procedur *BlockRead* i *BlockWrite*. Zastępują one odpowiednio procedury typem elementów. Pozostałe operacje jak *Assign*, *Reset*, *Rewrite*, *Close*, *Seek* i *Eof* mają taką samą interpretację jak dla plików elementowych.

Procedura *BlockRead*

Wywołanie: *BlockRead*(*FileVar*,*Buffer*,*Count*,*Reply*)
BlockRead(*FileVar*,*Buffer*,*Count*)

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej reprezentującą plik blokowy, *Buffer* jest nazwą dowolnej zmiennej programu, *Count* jest wyrażeniem typu *integer*, a *Reply* jest nazwą zmiennej typu *integer*.

Wykonanie procedury *BlockRead* powoduje wprowadzenie z pliku *FileVar*, do obszaru pamięci operacyjnej zajmowanego przez zmienną *Buffer*, *Count* bloków o rozmiarze 128 bajtów każdy. Jeśli posłużono się wywołaniem zawierającym argument *Reply*, to zmiennej reprezentowanej przez ten argument zostanie przypisana dana określająca liczbę faktycznie wprowadzonych bloków. Jeśli liczba ta jest mniejsza niż *Count*, to oznacza to, że plik znajduje się w pozycji końcowej.

Procedura *BlockWrite*

Wywołanie: *BlockWrite(FileVar,Buffer,Count,Reply)*
BlockWrite(FileVar,Buffer,Count)

Przyjmuje się, że *FileVar* jest nazwą zmiennej plikowej reprezentującą plik bez typu elementów, *Buffer* jest nazwą dowolnej zmiennej programu, *Count* jest wyrażeniem typu *integer*, a *Reply* jest nazwą zmiennej typu *integer*.

Wykonanie procedury *BlockWrite* powoduje wyprowadzenie do pliku *FileVar*, z obszaru pamięci zajmowanego przez zmienną *Buffer*, *Count* bloków o rozmiarze 128 bajtów każdy. Jeśli posłużono się wywołaniem zawierającym *Reply*, to zmiennej reprezentowanej przez ten argument zostanie przypisana dana określająca liczbę faktycznie wyprowadzonych bloków. Jeśli liczba ta jest mniejsza niż *Count*, to oznacza to, że wykonanie procedury nie przebiegło pomyślnie.

Przykład

```
Program Copy;
var
  Src,Trg : file;
  Buffer : array[0..255,boolean] of byte;
  Source,Target : string[40];
  Reply : integer;
begin
  Write('Source :— ');
  Readln(Source);
  Write('Target :— ')
  Readln(Target);
  Assign(Src,Source);
  Assign(Trg,Target);
  Reset(Src);
  Rewrite(Trg);
  repeat
    BlockRead(Src,Buffer,4,Reply);
    BlockWrite(Trg,Buffer,4,Reply)
  until Reply < 4;
  Close(Src);
  Close(Trg)
end.
```

• Wykonanie przytoczonego programu powoduje skopiowanie zbioru danych o dowolnym typie elementów. □

Kontrolowanie poprawności operacji wejścia/wyjścia

Sposób kontrolowania poprawności operacji wejścia/wyjścia jest uzależniony od sposobu skompilowania programu. W zakresie przyjmowanej przez domniemanie dyrektywy {$I+}, po każdej operacji wejścia/wyjścia jest kontrolowana jej poprawność. Jeśli zostanie wykryty błąd, to wykonywanie programu jest przerywane, a na konsolę jest wyprowadzany komunikat o rozpoznanym błędzie. W zakresie dyrektywy {$I−} rozpoznanie błędu nie powoduje wstrzymania wykonywania programu, a jedynie wstrzymanie wykonywania dalszych operacji wejścia/wyjścia. Stan taki utrzymuje się aż do momentu wywołania funkcji standardowej *IOresult*, udostępniającej daną typu integer. Jeśli rezultatem wywołania tej funkcji jest dana o wartości 0, to oznacza to, że dotychczasowe operacje wejścia/wyjścia zostały wykonane poprawnie. Jeśli rezultatem jest dana o wartości różnej od zera, to wartość ta określa rodzaj rozpoznanego błędu. Wykaz błędów operacji wejścia/wyjścia przytoczono w dodatku C.

Przykład

```
program Delete;
var
  FileVar : file;
  FileName : string[40];
  Flag : boolean;
begin
  Write('FileName:= ');
  Readln(FileName);
  Assign(FileVar,FileName);
  {$I−} Erase (FileVar); {$I+}
    if not (IOresult = 0) then
    Writeln('File'+FileName+' did not exist');
end.
```

• Wykonanie przytoczonego programu powoduje usunięcie zbioru danych o nazwie wprowadzonej z konsoli.

• Jeśli podano nazwę zbioru nie istniejącego, to zostanie wyprowadzony komunikat, że zbiór ten nie istniał.

• Gdyby z programu usunięto dyrektywy kompilatora, a zbiór by nie istniał, to wykonywanie programu zostałoby przerwane na skutek wystąpienia błędu operacji *Erase* dotyczącej zbioru, który nie istnieje. □

13. Typy wskazujące

Każdy z typów wskazujących jest typem prostym. Wartościami danych wskazujących są wskazania danych.

Opis typu wskazującego składa się ze znaku ^(caret), bezpośrednio po którym występuje nazwa typu. Nie wymaga się, aby ta nazwa była już zdefiniowana. Jest to jedyny wyjątek od wymagania powoływania się na obiekty uprzednio zdefiniowane.

Składnia

opis-typu-wskazującego:
  ^*nazwa-typu-bazowego*
nazwa-typu:
  identyfikator

Przykłady

```
type
  EmployeePtr = ^EmployeeData;
  EmployeeData = record
                   Name : string[30];
                   Salary : real
                 end;
var
  EmployeeRef : EmployeePtr;
  IntRef : ^integer;
```

• *Typ EmployeePtr* jest związany ze zbiorem wskazań zmiennych typu *EmployeeData*.
• *EmployeeRef* jest nazwą zmiennej wskazującej. Zmiennej tej mogą być przypisywane dane wskazujące zmienne typu *EmployeeData*.

• *IntRef* jest nazwą zmiennej wskazującej. Zmiennej tej mogą być przypisywane dane wskazujące zmienne typu *integer*. □

Poza typami i zmiennymi wskazującymi występuje w Turbo Pascalu wskazanie puste reprezentowane przez słowo kluczowe **nil**. Typ tego wskazania jest zgodny z dowolnym innym typem wskazującym. Nazwy zmiennych wskazujących oraz wskazanie puste mogą występować w relacjach = (równe) i < > (nie równe). W instrukcji przypisania wymaga się zgodności typów lewej i prawej strony. Zgodność ta ma miejsce tylko wtedy, gdy prawa strona jest słowem kluczowym **nil** oraz wtedy, gdy prawa strona reprezentuje daną wskazującą takiego samego typu jak dane, które mogą być przypisywane zmiennej występującej z lewej strony dwuznaku przypisania.

Przykład

```
type
  Days = (Mon,Tue,Wed,Thu,Fri,Sat,Sun);
  Week = Mon..Sun;
var
  Day1 : ^Days;
  Day2,Day3 : ^Week;
```

• W zasięgu przytoczonych definicji i deklaracji poprawne są m.in. przypisania

```
Day1 := nil;
Day2 := Day3
```

ale nie jest poprawne przypisanie

```
Day2 := Day1;
```

ponieważ zmienne *Day2* i *Day1* są różnych typów. □

Zmienne wskazujące mogą zostać wykorzystane do dynamicznego zarządzania pamięcią operacyjną. Pomocne są w tym procedury do przydzielania i zwalniania tej pamięci.

Procedura *New*

Wywołanie: *New(PtrVar)*

Przyjmuje się, że *PtrVar* jest nazwą zmiennej wskazującej. Wykonanie procedury *New* powoduje utworzenie zmiennej takiego samego typu, jakiego są zmienne, które mogą być wskazywane przez dane przypisywane zmiennej *PtrVar*. Po wykonaniu tej czynności zmiennej *PtrVar* zostaje przypisana dana wskazująca właśnie utworzoną zmienną. Wymieniona tu zmienna zostaje utworzona w obszarze pamięci operacyjnej nazywanym stertą i może być traktowana tak, jak dowolna inna zmienna jej typu.

Przykład

```
type
  Matrix = array[boolean] of array[boolean] of real;
var
  MatrixPtr : ^Matrix;
begin
  New(MatrixPtr);
  MatrixPtr^ [true] [true] := 23.5;
end.
```

- Wykonanie procedury *New* powoduje utworzenie na stercie zmiennej typu *Matrix*, a następnie przypisanie zmiennej wskazującej *MatrixPtr*, danej wskazującej tę zmienną.
- Wykonanie instrukcji przypisania powoduje przypisanie narożnikowemu elementowi tablicy *MatrixPtr^* danej o wartości 23.5.
- Należy zauważyć, że napis *MatrixPtr* jest nazwą zmiennej wskazującej, a napis *MatrixPtr^* jest nazwą obiektu wskazywanego przez tę zmienną.

Procedura *Dispose*

Wywołanie: *Dispose*(*PtrVar*)

Przyjmuje się, że *PtrVar* jest nazwą zmiennej wskazującej.

Wykonanie procedury *Dispose* powoduje eliminację zmiennej wskazywanej przez daną przypisaną zmiennej *PtrVar*. Wymaga się, aby eliminowana zmienna była uprzednio utworzona za pomocą procedury *New*. Obszar pamięci operacyjnej, zajmowany przez eliminowaną zmienną zostaje zwrócony do sterty i może być wykorzystany do tworzenia innych zmiennych.

Przykład

```
var
  IntRef : ^integer;
begin
  New(IntRef);
  IntRef^ := 20;
  Dispose(IntRef);
  New(IntRef);
  IntRef^ := 30;
  Writeln(IntRef^)
end.
```

- Wykonanie przytoczonego programu powoduje wyprowadzenie napisu 30.
- Gdyby z programu usunięto drugą instrukcję przypisania, to byłby on

błędny, ponieważ w momencie wykonywania procedury *Writeln* zmiennej *IntRef* nie byłaby przypisana żadna dana. □

Procedura *Mark*

Wywołanie: *Mark(PtrVar)*

Przyjmuje się, że *PtrVar* jest nazwą zmiennej wskazującej dowolnego typu.

Wykonanie procedury *Mark* powoduje przypisanie zmiennej *PtrVar* wskazania bieżącego szczytu sterty.

Procedura *Release*

Wywołanie: *Release(PtrVar)*

Przyjmuje się, że *PtrVar* jest nazwą zmiennej wskazującej dowolnego typu.

Wykonanie procedury *Release* powoduje usunięcie ze sterty zmiennej wskazywanej przez *PtrVar* i wszystkich zmiennych następujących po niej na stercie.

Procedury *New-Dispose* oraz *Mark-Release* dostarczają dwóch mechanizmów zarządzania pamięcią sterty. W ustalonym programie należy ograniczyć się do jednego z tych mechanizmów. Istota różnic między rozpatrywanymi tu

Tablica 13.1. Porównanie mechanizmów zarządzania pamięcią sterty

Sterta	Po *Dispose*	Po *Release*
var 1	var 1	var 1
var 2	var 2	var 2
var 3		
var 4	var 4	
var 5	var 5	

mechanizmami została przedstawiona w tabl. 13.1. Przy założeniu, że zmiennej *Ptr* przypisano wskazanie zmiennej *var3*, pokazano tam skutek wykonania procedury *Dispose(Ptr)* i *Release(Ptr)*.

Procedura *GetMem*

Wywołanie: *GetMem(PtrVar,IntExp)*

Przyjmuje się, że *PtrVar* jest nazwą zmiennej wskazującej a *IntExp* jest wyrażeniem typu *integer*.

Wykonanie procedury *GetMem* powoduje zarezerwowanie na stercie obszaru pamięci operacyjnej o rozmiarze wyrażonym w bajtach i określonym przez wartość wyrażenia *IntExp*, a następnie przypisanie zmiennej *PtrVar* danej wskazującej tak przypisany obszar.

Procedura *FreeMem*

Wywołanie: *FreeMem(PtrVar,IntExp)*

Przyjmuje się, że PtrVar jest nazwą zmiennej wskazującej, a *IntExp* jest wyrażeniem typu *integer.*

Wykonanie procedury *FreeMem* powoduje zwrócenie do sterty obszaru pamięci operacyjnej wskazywanego przez zmienną *PtrVar*, zajmującego tyle bajtów, ile określa wartość danej reprezentowanej przez wyrażenie *IntExp.* Wymaga się, aby obszar zwracany za pomocą procedury *FreeMem* miał dokładnie taki sam rozmiar jak obszar uzyskany za pomocą *GetMem.*

Funkcja *MaxAvail*

Wywołanie: *MaxAvail*

Rezultatem funkcji *MaxAvail* jest dana typu *integer* określającego rozmiar największego spójnego obszaru pamięci operacyjnej dostępnego na stercie. Rozmiar ten jest wyrażony liczbą *paragrafów.* Każdy paragraf liczy 16 bajtów. Jeśli rezultat jest ujemny, to faktyczną liczbę paragrafów można uzyskać dodając do rezultatu liczbę rzeczywistą o wartości 65536.

Przykład

```
type
  Pointer = ^Pair;
  Pair : record
           Int : integer;
           Ref : Pointer
         end
var
  Number : integer;
  Head,Tail : Pointer;
begin
  Head := nil;
  Read(Number);
  while not Eof do begin
    New(Tail);
    Tail^.Int := Number;
    Tail^.Ref := Head;
```

```
      Head := Tail;
      Read(Number)
    end;
    Tail := Head;
    while Tail <> nil do begin
      Writeln(Tail^.Int);
      Tail := Tail^.Ref
    end
end.
```

• Wykonanie przytoczonego programu powoduje wprowadzenie z konsoli ciągu liczb całkowitych, a następnie wyprowadzenie go w kolejności odwrotnej. □

14. Przypisywanie danych początkowych

Jedną z ważnych właściwości języka Turbo Pascal jest możliwość *przypisywania zmiennym danych początkowych*. Jeśli podczas wykonywania programu zmienne takie zachowują swoje początkowe wartości, to mogą być traktowane jak stałe. Jeśli natomiast zmienna, której przypisano daną początkową zmieni swoją wartość, to w chwili ponownego aktywowania programu, znajdującego się w pamięci operacyjnej, zostanie zachowana ostatnia wartość zmiennej z poprzedniego wykonania. Ponieważ taki program z reguły traci właściwość powtarzalności, zaleca się, aby przypisywanie danych początkowych było ograniczone do zmiennych zachowujących się jak stałe. Z tej to właśnie przyczyny, deklaracje zmiennych, którym należy przypisać dane początkowe, występują w programie po słowie kluczowym **const**, a nie po słowie **var**.

Przykład

```
program Increment;
const
  Number : integer = 1;
begin
  Writeln('Number = ',Number);
  Number:= Number+1
end.
```

• W przytoczonym programie zmiennej *Number* typu *integer* przypisano daną początkową o wartości 1.
• Kolejne aktywowanie programu za pomocą dyrektywy R będzie powodować wyprowadzanie kolejnych liczb naturalnych. □

Jak już częściowo wynika z przytoczonego przykładu, deklaracja zmiennej, której przypisano daną początkową, składa się z nazwy zmiennej, po której następuje dwukropek, następnie opisu typu zmiennej, znak równości, *inicjator*

oraz średnik. Zapis inicjatora zależy od typu zmiennej. Dla zmiennych prostych ma on postać literału, natomiast dla zmiennych tablicowych ma postać listy inicjatorów ujętych w nawiasy okrągłe, a dla zmiennych rekordowych postać wykazu inicjatorów, także ujętego w nawiasy okrągłe.

Składnia

deklaracja-z-inicjacją:
  nazwa-zmiennej : *opis-typu* = *inicjator* ;
nazwa-zmiennej:
  identyfikator
inicjator:
  literal
  nazwa-literalu
  (*lista-inicjatorów*)
  (*wykaz-inicjatorów*)

Przykłady

const
  LineLength : *byte* = 132;
  Radius : *real* = 13.64;
  FullName : **string**[12] = '*Jan Bielecki*';
  CtrlM : *char* = ^*M*;
  Separator : **set of** *char* = ['/', ';' ,'.'];
  SexIsMale : *boolean* = *true*;
  ShortName : **string**[3] = '*Izabela*';
  WholeReal : *real* = 44;

- Zmiennej *LineLength* (typu *byte*) przypisano daną początkową 132.
- Zmiennej *Radius* (typu real) przypisano daną początkową 13.64.
- Zmiennej *FullName* (typu **string**[12]) przypisano daną '*Jan Bielecki*'.
- Zmiennej *CtrlM* (typu *char*) przypisano daną ^*M*.
- Zmiennej *Separator*, (typu **set of** *char*) przypisano daną ['/', ';' ,'.'].
- Zmiennej *SexIsMale* (typu *boolean*) przypisano daną *true*.
- Zmiennej *ShortName* (typu **string**[3]) przypisano daną '*Iza*'.
- Zmiennej *WholeReal* przypisano daną 44.0. □

Jeśli inicjowana zmienna jest tablicą, to inicjator musi mieć postać listy ujętej w nawiasy okrągłe, wyszczególniającej dane początkowe dla wszystkich elementów tej tablicy. Jeśli inicjowana zmienna jest rekordem, to inicjator musi mieć postać wykazu ujętego w nawiasy okrągłe, wyszczególniającego dane początkowe dla wszystkich albo tylko niektórych pól rekordu. Wymaga się, aby elementy wykazu wyszczególniały dane początkowe w takiej kolejności, w jakiej pola rekordu występują w jego deklaracji. W przypadku inicjowania

rekordów elementami wykazu są oddzielone średnikami napisy składające się z nazwy pola rekordu, po której następuje dwukropek i inicjator pola rekordu.

Przykłady

```
type
  Color = (Red,Black,Fair);
  Person = record
             FirstName : string[3]
             LastName : string[12];
             Hair : Color;
             Age : byte
           end;
  Sex = (male,female);
const
  Vector : array[1..3] of Sex =
           (male,male,female);
  Matrix : array[Color,2..3] of byte =
           ((0,0),(1,1),(2,5));
  ThreeD : array[boolean,boolean,boolean] of Color =
           (((Red,Red),(Red,Fair)),
           ((Black,Black),(Red,Black)));
  JanB : Person =
         (FirstName : 'Jan';
         LastName : 'Bielecki';
         Hair : Black;
         Age : 44);
  Family : array[1..3] of Person =
           ((FirstName : 'Jan';
           LastName : 'Bielecki';
           Hair : Black;
           Age : 44),
           (FirstName : 'Ewa';
           LastName : 'Bielecka';
           Hair : Black;
           Age : 38),
           (FirstName : 'Iza';
           LastName : 'Bielecka';
           Hair : Black;
           Age : 3));
```

- Elementowi *Vector*[2] przypisano daną początkową *male*.
- Elementowi *Matrix*[*Fair*,3] przypisano daną początkową 5.
- Elementowi *ThreeD*[*false*,*true*,*true*] przypisano daną początkową *Fair*.

- Polu *JanB.Age* przypisano daną początkową 44.
- Polu *Family*[2].Age przypisano daną początkową 38.
- Gdyby deklaracji zmiennej *Vector* nadano postać

Vector : **array**[1..3] **of** *Sex* = (*male,female*);

to byłaby ona błędna, ponieważ zmienna ta jest tablicą 3-elementową, natomiast lista inicjatorów zawarta w nawiasach okrągłych składa się z tylko 2 elementów.

- Gdyby deklaracji zmiennej *JanB* nadano postać

JanB : *Person* = (*FirstName* : '*Janek*');

to byłaby ona poprawna i równoważna deklaracji

JanB : *Person* = (*FirstName* : '*Jan*');

- Gdyby deklaracji zmiennej *JanB* nadano postać

JanB : *Person* = (*LastName* : '*Bielecki*');

to byłaby ona niepoprawna. □

Przypisania danych początkowych polom rekordu z wariantami są sensowne tylko wtedy, gdy dotyczą tych pól wariantu, które do niego należą. Wymaganie to nie jest kontrolowane przez kompilator.

Przykład

```
type
  Sex = (male,female);
  Color = (Black,Red,Blond);
  Person = record
             Name : string[20];
             case Gender : Sex of
               male : (Height : byte);
               female : (Hair : Color)
           end;
const
  Trio : array[1..3] of Person = (
           (Name : 'Ewa'),
           (Name : 'Jan'; Gender : male),
           (Name : 'Iza'; Gender : female; Hair : Black));
```

- Polu *Trio*[2].*Gender* tablicy rekordów *Trio* przypisano daną początkową *male*.
- Inicjator o postaci

(*Name* : '*Iza*'; *Gender* : *male*; *Hair* : *Black*)

chociaż mało sensowny, jest uznawany za poprawny. □

15. Funkcje i procedury

Funkcje i procedury są obiektami opisującymi dobrze zdefiniowane fragmenty algorytmu realizowanego przez program. Z tego względu będą nazywane *podprogramami.*

W odróżnieniu od innych obiektów strukturalnych, jak np. instrukcji wyboru, wykonanie podprogramu wymaga jego *wywołania*, tj. stosownie do sytuacji, posłużenia się instrukcją wywołania procedury albo wywołaniem funkcji. *Instrukcja wywołania procedury* może wystąpić w każdym miejscu programu, w którym może być użyta np. instrukcja pusta, natomiast *wywołanie funkcji* może być użyte jedynie w wyrażeniu, występując tam pod postacią tzw. *nazewnika funkcji.*

Podprogramy, podobnie jak zmienne, wymagają deklarowania. Deklaracje podprogramów mogą być umieszczane w części deklaracyjnej bloku. *Deklaracja podprogramu*, nazywana niekiedy jego *definicją*, składa się z *nagłówka* oraz z *bloku* stanowiącego *ciało podprogramu. Nagłówek procedury* składa się ze słowa kluczowego **procedure**, po którym następuje nazwa procedury, ujęty w nawiasy okrągłe wykaz parametrów procedury oraz średnik. *Nagłówek funkcji* składa się ze słowa kluczowego **function**, po którym następuje nazwa funkcji, ujęty w nawiasy okrągłe wykaz parametrów funkcji, dwukropek, określenie typu rezultatu funkcji oraz średnik. Jeśli wykaz parametrów podprogramu jest pusty, to jest opuszczany wraz z otaczającymi go nawiasami.

Wymaga się, aby podczas wykonywania funkcji została wykonana instrukcja przypisania, w której z lewej strony dwuznaku przypisania występuje nazwa funkcji, a z prawej wyrażenie zgodne z typem rezultatu funkcji. Dana reprezentowana przez wyrażenie, które wystąpiło w ostatnim tak wykonanym przypisaniu stanowi rezultat funkcji. Wykonanie funkcji nie musi ograniczyć się do udostępnienia rezultatu. Może ono także spowodować zmianę pewnych argumentów jej wywołania.

Składnia

deklaracja-podprogramu:
deklaracja-procedury
deklaracja-funkcji

deklaracja-procedury:
procedure *nazwa-procedury* (*wykaz-parametrów*) ;
procedure *nazwa-procedury* ;

deklaracja-funkcji:
function *nazwa-funkcji* (*wykaz parametrów*) : *typ-rezultatu* ;
function *nazwa-funkcji* : *typ-rezultatu* ;

nazwa-procedury:
identyfikator

nazwa-funkcji:
identyfikator

typ-rezultatu:
identyfikator-typu-prostego

identyfikator-typu-prostego:
identyfikator

Elementy wykazu parametrów podprogramu są oddzielone średnikami. Każdy element wykazu zawiera listę identyfikatorów parametrów, po której następuje dwukropek i identyfikator typu parametrów danej listy.

W chwili wywołania podprogramu następuje skojarzenie parametrów podprogramu z argumentami jego wywołania. Liczba argumentów wywołania musi być równa liczbie parametrów, a skojarzenia parametrów z argumentami dokonują się w kolejności ich wystąpienia w nagłówku podprogramu i w wywołaniu.

Skojarzenie parametru z argumentem może dokonać się przez wartość albo przez wskazanie. W pierwszym przypadku parametr jest traktowany jak lokalna zmienna podprogramu, której w chwili rozpczęcia jego wykonywania (dla danego wywołania) przypisano daną reprezentowaną przez argument. W drugim przypadku parametr jest traktowany tak, jakby reprezentował argument. Ma to taki skutek, że każda operacja dotycząca parametru jest realizowana tak, jakby dotyczyła argumentu. Posłużenie się tym rodzajem skojarzenia parametru z argumentem wymaga poprzedzenia listy identyfikatorów parametrów słowem kluczowym **var**.

Szczególnym rodzajem skojarzenia przez wskazanie jest skojarzenie parametru z argumentem, który jest nazwą podprogramu. Jednym z ograniczeń języka Turbo Pascal w stosunku do standardu języka Pascal jest brak tego rodzaju skojarzenia.

Składnia

element-wykazu-parametrów:
lista-nazw-parametrów : oznaczenie-typu
var *lista-nazw-parametrów : oznaczenie-typu*
var *list-nazw-parametrów*
oznaczenie-typu:
identyfikator-typu
identyfikator-typu:
identyfikator
nazwa-parametru:
identyfikator

Przykłady

a. Procedura z jednym parametrem, skojarzenie przez wartość

```
program FiveMessages;
var
  Tally : byte;
procedure SlowDown(Count : integer);
begin
  for Count:= Count downto 1 do
end;
begin
  for Tally:= 1 to 5 do begin
    Writeln('Wake up');
    SlowDown(400)
  end
end.
```

- Program *FiveMessages* zawiera 2 deklaracje: deklarację zmiennej *Tally* i deklarację procedury *SlowDown*.
- Wykazem, a zarazem elementem wykazu parametrów, jest napis

Count : integer

- Skojarzenie parametru *Count* z argumentem 400 dokona się przez wartość.
- Operacje na parametrze *Count* są w istocie operacjami na pewnej lokalnej zmiennej procedury *SlowDown*. Początkową wartością tej zmiennej jest 400.

b. Procedura z dwoma parametrami, skojarzenie przez wskazanie

```
program ConvertAndSwap;
var
  theFloat : real;
  theFixed : integer;
```

```
procedure Swap(var Float : real;
               var Fixed : integer);
var
  Temp : real;
begin
  Temp := Float;
  Float := Fixed;
  Fixed := trunc(Temp+0.5)
end;
begin
  theFloat := 12.8;
  theFixed := 10:
  Writeln(theFloat,theFixed :3);
  Swap(theFloat,theFixed);
  Writeln(theFloat,theFixed :3)
end.
```

• Program *ConvertAndSwap* zawiera 4 deklaracje: deklaracje zmiennych *theFloat* i *theFixed*, deklarację procedury *Swap* oraz deklarację zmiennej *Temp*.

• Wykaz parametrów procedury *Swap* składa się z 2 elementów oddzielonych średnikiem.

• Skojarzenia parametrów procedury z argumentami wywołania dokonują się przez wskazanie.

• Operacje na parametrze *Float* są w istocie operacjami na argumencie theFloat, a operacje na parametrze *Fixed* są operacjami na argumencie *theFixed*.

• Wykonanie programu powoduje wyprowadzenie 2 wierszy tekstu. W pierwszym znajdą się liczby 12.8 i 10, a w drugim liczby 10 i 13.

• Gdyby nagłówkowi procedury *Swap* nadano postać

```
procedure Swap(Float : real;
               var Fixed : integer);
```

niczego poza tym nie zmieniając, to zostałyby wyprowadzone liczby 12.8 i 10 oraz 12.8 i 13. Wynika to z faktu, że po wprowadzeniu takiej zmiany skojarzenie parametru *Float* z argumentem *theFloat* dokonałoby się przez wartość.

c. Najkrótszy program zawierający deklarację i wywołanie procedury bezparametrowej

```
procedure P;
begin
end;
begin
  P
end.
```

d. Funkcja z trzema parametrami

```
program MultiplyDivide;
var
  Product : integer;
function Divide(SourceOne,SourceTwo : byte;
                var Target : integer) : real;
begin
  Target := SourceOne*SourceTwo;
  Divide := SourceOne/SourceTwo
end;
begin
  Writeln(Divide(12,4,Product),Product :3)
end.
```

• Parametry *SourceOne* i *SourceTwo* są skojarzone z argumentami 12 i 4 przez wartość. Parametr *Target* jest skojarzony z argumentem *Product* przez wskazanie. Rezultatem funkcji jest dana typu *real*.
• Wykonanie programu powoduje wyprowadzenie 1 wiersza tekstu zawierającego liczby 3.0 i 48.
• Gdyby nagłówkowi funkcji *Divide* nadano postać

```
function Divide(SourceOne,SourceTwo : byte;
                Target : integer) : real;
```

to program byłby niepoprawny z powodu nieprzypisania zmiennej *Product* żadnej danej.
• Gdyby nagłówkowi funkcji *Divide* nadano natomiast postać

```
function Divide(var SourceOne,SourceTwo : byte;
                var Target : integer) : real;
```

to program byłby niepoprawny, ponieważ z parametrami zadeklarowanymi do skojarzenia przez nazwę próbowano by skojarzyć argumenty nie będące nazwami zmiennych, lecz wyrażeniami, tu 12 i 4.

e. Najkrótszy program zawierający deklaracje i wywołanie funkcji bezparametrowej

```
function F : byte;
begin
  F := 0
end;
begin
  Write(F)
end.
```

□

• Przytoczony program zawiera konstrukcje języka standardowego, które nie zostały implementowane w języku Turbo Pascal. Z tego powodu jest to program błędny. □

Kontrolowane przez kompilator wymaganie zgodności typu parametru z typem argumentu nie dotyczy tzw. *parametrów bez typu*, zadeklarowanych bez opisu typu. Przyjmuje się bowiem, że nie ujawniony typ takiego parametru jest zgodny z typem dowolnego, skojarzonego z nim argumentu. To rozszerzenie języka, często w połączeniu z użyciem opisanego w rozdz. 19 słowa kluczowego **absolute** umożliwia w wielu przypadkach opracowywanie bardziej efektywnych programów.

Przykład

```
program MoveArray;
var
  SourceArray : array[1..3,1..4] of integer;
  TargetArray : array[1..2,1..6] of integer;
...
procedure MoveBytes(var Source,Target;
                    Count : integer);
var
  Index : integer;
type
  ByteField = array[1..MaxInt] of byte;
var
  Src : ByteField absolute Source;
  Trg : ByteField absolute Target;
begin
  for Index:= 1 to Count do
    Trg[Index]:= Src[Index]
end;
...
begin
  ...
  MoveBytes(SourceArray,TargetArray,24)
  ...
end.
```

• Tablice *SourceArray* i *TargetArray* są różnych typów, zatem wykonanie instrukcji

TargetArray:= *SourceArray*

jest zabronione. Rezultat ten można jednak uzyskać, posługując się procedurą *MoveBytes*.

• W procedurze tej *Source* i *Target* są parametrami bez typu. Lokalne zmienne *Src* i *Trg* typu *ByteField* zostają odwzorowane na obszary pamięci zajmowane przez tablice skojarzone z tymi parametrami. □

Program, w którym parametr jawnie określonego typu jest skojarzony z argumentem innego typu, jest błędny. Wymaganie zgodności typu parametru i argumentu może zostać osłabione w odniesieniu do parametrów i argumentów typu łańcuchowego. Przyjęto bowiem, że w zasięgu działania dyrektywy kompilatora {$V—} każdy parametr typu łańcuchowego jest zgodny ze skojarzonym z nim argumentem typu łańcuchowego. Umożliwia to konstruowanie procedur do przetwarzania danych łańcuchowych dowolnych typów.

Przykład

```
program CountSpaces;
type
  OneLine = string[80];
function SpaceCount(SourceText : OneLine) : integer;
var
  Count,Index : integer;
begin
  Count := 0;
for Index := 1 to Length(SourceText) do
    if SourceText[Index] = '' then
      Count := Count+1;
  SpaceCount := Count
end;
begin
  ...
  Writeln(SpaceCount(
    {$V—} 'To be or not to be' {$V+}
                    ));
  ...
end.
```

• Parametr *SourceText* jest typu **string**[80], natomiast skojarzony z nim argument jest typu **string**[18].
• Wykonanie instrukcji wywołania procedury *Writeln* powoduje wyprowa-
metru z argumentem odbywa się w zasięgu dyrektywy kompilatora {$V—}.
• Wykonanie instrukcji wywołania procedury *Writeln* powoduje wyprowadzenie liczby 5. □

Podobnie jak w wielu innych językach algolopodobnych, podprogramy

zapisane w języku Turbo Pascal mogą zostać przygotowane w wersji rekurencyjnej. Niejednokrotnie umożliwia to uproszczenie algorytmu.

Przykład

```
function Fibonacci(Index : byte) : integer;
begin
  if Index < 3 then
    Fibonacci:= 1
  else
    Fibonacci:= Fibonacci(Index—1)+
                Fibonacci(Index—2)
end;
```

• Przytoczona funkcja służy do wyznaczania wartości wyrazów ciągu Fibonacciego. W ciągu tym pierwsze dwa wyrazy mają wartość 1, natomiast każdy z następnych jest sumą dwóch bezpośrednio go poprzedzających. □

W tych przypadkach, gdy posłużenie się rekurencją wymaga zadeklarowania np. dwóch procedur, z których pierwsza wywołuje drugą, a druga wywołuje pierwszą, niemożliwe jest takie uporządkowanie deklaracji, aby każde odwołanie dotyczyło obiektu już zadeklarowanego. W takiej sytuacji konieczne jest posłużenie się *deklaracją zapowiadającą*, składającą się z nagłówka podprogramu, po którym następuje średnik i słowo kluczowe **forward**.

W dogodnym miejscu po deklaracji zapowiadającej powinna wystąpić właściwa deklaracja podprogramu. Jej nagłówek nie zawiera już wykazu parametrów ujętego w nawiasy, a dla funkcji — także dwukropka i następującego po nim oznaczenia typu.

Przykład

```
function FunOne(var Source : byte; Target : byte) : char;
  forward;
function FunTwo(Count : byte) : real;
begin
  ...
  Write(FunOne(Count,2*Count));
  ...
end;
function FunOne;
begin
  ...
  Write(FunTwo(Source+Target));
  ...
end;
```

• Ponieważ w ciele definicji funkcji *FunTwo* wystąpiło odwołanie do funkcji *FunOne*, a w ciele funkcji *FunOne* wystąpiło odwołanie do funkcji *FunTwo*, konieczne było posłużenie się deklaracją zapowiadającą, w której ciało funkcji *FunOne* zostało zastąpione napisem **forward**.

• Oczywiście nic nie stałoby na przeszkodzie, aby deklaracje przytoczonych funkcji wystąpiły w odwrotnej kolejności. W takim przypadku deklaracja zapowiadająca dotyczyłaby funkcji *FunTwo* i miałaby postać

function *FunTwo*(*Count* : *byte*) : *real*; □

16. Podprogramy standardowe

Określenie podprogram standardowy dotyczy tych podprogramów, które są dostępne bez jawnego definiowania. Wiele takich podprogramów opisano już uprzednio. Należą do nich podprogramy służące do wykonywania operacji na danych łańcuchowych, podprogramy do dynamicznego zarządzania pamięcią operacyjną oraz podprogramy realizujące operacje wejścia/wyjścia.

Uzupełnieniem tego obszernego zbioru podprogramów są procedury ekranowe i specjalne, funkcje arytmetyczne, skalarne i do wykonywania konwersji oraz dość liczna grupa funkcji pomocniczych.

Procedury ekranowe

Procedury ekranowe są dostępne w tych implementacjach, w których dokonano ich instalacji.

Procedura *ClrEol*

Wywołanie: *ClrEol*

Procedura *ClrEol* jest bezparametrowa.

Wykonanie procedury *ClrEol* powoduje zlokalizowanie wiersza, w którym znajduje się kursor, a następnie zastąpienie spacjami wszystkich znaków tego wiersza, począwszy od znaku wyróżnionego przez kursor. Pozycja kursora nie ulega zmianie.

Procedura *ClrScr*

Wywołanie: *ClrScr*

Procedura *ClrScr* jest bezparametrowa.

Wykonanie procedury *ClrScr* powoduje tzw. „wyczyszczenie" ekranu, tj. zapełnienie go znakami spacji. Kursor zostanie umieszczony w lewym-górnym rogu ekranu. W zależności od sposobu zainstalowania procedury czynności tej może towarzyszyć zmiana parametrów ustalonych za pomocą opisanych dalej procedur *LowVideo* i *NormVideo.*

Procedura *CrtInit*

Wywołanie: *CrtInit*

Procedura *CrtInit* jest bezparametrowa.

Wykonanie procedury *CrtInit* powoduje skierowanie do monitora łańcucha znaków inicjujących, określonego podczas instalowania systemu Turbo Pascal.

Procedura *CrtExit*

Wywołanie: *CrtExit*

Procedura *CrtExit* jest bezparametrowa.

Wykonanie procedury *CrtExit* powoduje skierowanie do monitora łańcucha znaków kończących, określonego podczas instalowania systemu Turbo Pascal.

Procedura *DelLine*

Wywołanie: *DelLine*

Procedura *DelLine* jest bezparametrowa.

Wykonanie procedury *DelLine* powoduje zlokalizowanie wiersza, w którym znajduje się kursor, a następnie usunięcie go i przemieszczenie wierszy znajdujących się poniżej o jeden wiersz do góry. Po wykonaniu tych czynności najniższy wiersz ekranu staje się pusty. Pozycja kursora nie ulega zmianie.

Procedura *InsLine*

Wywołanie: *InsLine*

Procedura *InsLine* jest bezparametrowa.

Wykonanie procedury *InsLine* powoduje zlokalizowanie wiersza, w którym znajduje się kursor, a następnie przemieszczenie tego wiersza, wraz z wierszami za nim następującymi, o jeden wiersz do dołu. Powoduje to umieszczenie na ekranie pustego wiersza i usunięcie z ekranu najniżej położonego z przemieszczanych wierszy. Pozycja kursora nie ulega zmianie.

Procedura *GotoXY*

Wywołanie: *GotoXY(xPos,yPos)*

Przyjmuje się, że *xPos* i *yPos* są wyrażeniami typu *integer*.

Wykonanie procedury *GotoXY* powoduje ustawienie kursorsa w takiej pozycji, że wyróżnia on znak o współrzędnych (*xPos,yPos*). Zakłada się, że lewy-górny narożnik ekranu ma współrzędne (1,1), odcięte są liczone od lewej do prawej, a rzędne od góry do dołu. Jeśli wartość wyrażenia *xPos* albo *yPos* wykracza poza rozmiar wiersza albo kolumny, to każda z nich jest traktowana modulo rozmiar.

Procedura *LowVideo*

Wywołanie: *LowVideo*

Procedura *LowVideo* jest bezparametrowa.

Wykonanie procedury *LowVideo* powoduje, że aż do najbliższego wykonania procedury *NormVideo* znaki wyprowadzone na ekran są przyciemnione albo poddane inwersji.

Procedura *NormVideo*

Wywołanie: *NormVideo*

Procedura *NormVideo* jest bezparametrowa.

Wykonanie procedury *NormVideo* powoduje przywrócenie normalnego (por. procedura *LowVideo*) sposobu wyświetlania znaków.

Procedury specjalne

Procedura *Delay*

Wywołanie: *Delay(Time)*

Przyjmuje się, że *Time* jest wyrażeniem typu *integer*.

Wykonanie procedury *Delay* powoduje wstrzymanie wykonywania programu na okres (w przybliżeniu) *Time* milisekund. Jeśli wyrażenie *Time* ma wartość ujemną, to traktuje się je tak, jakby miało wartość 0.

Procedura *FillChar*

Wywołanie: *FillChar(Var,Cnt,Val)*

Przyjmuje się, że *Var* jest nazwą zmiennej dowolnego typu, *Cnt* jest wyrażeniem typu *integer*, a *Val* jest wyrażeniem typu *char* albo *byte*.

Wykonanie procedury *FillChar* powoduje umieszczenie w obszarze, którego początek przydzielono zmiennej *Var*, *Cnt* danych bajtowych mających taką samą reprezentację jak dana bajtowa reprezentowana przez *Val*.

Procedura *Exit*

Wywołanie: *Exit*

Procedura *Exit* jest bezparametrowa.

Wykonanie procedury *Exit* ma taki sam skutek, jak wykonanie instrukcji przejścia do niejawnej instrukcji pustej poprzedzającej słowo kluczowe **end** tego bloku, w którym wywołanie to wystąpiło. Powoduje to zatem zakończenie wykonywania bloku, a jeśli jest nim blok programu, także zakończenie wykonywania całego programu.

Procedura *Halt*

Wywołanie: *Halt*
Halt(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*. W systemie CP/M jest dostępna jedynie bezparametrowa wersja procedury *Halt*. W systemie DOS wywołanie *Halt* jest równoważne wywołaniu *Halt*(0).

Wykonanie procedury *Halt* powoduje zakończenie wykonywania programu. W systemie DOS wartość jawnego albo domniemanego argumentu jest przekazywana systemowi operacyjnemu. Wartość ta może zostać wykorzystana np. do wykonania testu ERRORLEVEL.

Procedura *Move*

Wywołanie: *Move*(*Src*,*Trg*,*Cnt*)

Przyjmuje się, że *Src* i *Trg* są nazwami zmiennych dowolnego typu, a *Cnt* jest wyrażeniem typu *integer*.

Wykonanie procedury *Move* powoduje przepisanie z obszaru, którego początek przydzielono zmiennej *Src*, do obszaru, którego początek przydzielono zmiennej *Trg*, *Cnt* bajtów pamięci. Przepisanie jest poprawne nawet wtedy, gdy wspomniane obszary częściowo albo w całości pokrywają się.

Procedura *Randomize*

Wywołanie: *Randomize*

Procedura *Randomize* jest bezparametrowa.

Wykonanie procedury *Randomize* powoduje zainicjowanie generatora liczb przypadkowych daną o wartości przypadkowej.

Funkcje arytmetyczne

Funkcja *Abs*

Wywołanie: *Abs*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real* albo *integer*.

Rezultatem funkcji *Abs* jest dana takiego samego typu jak typ argumentu, o wartości równej wartości bezwzględnej danej reprezentowanej przez Num.

Przykładowo: *Abs*(−2) = 2
Abs(*Pi*) = 3.1415926536

Funkcja *ArcTan*

Wywołanie: *ArcTan*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real* albo *integer*.

Rezultatem funkcji *ArcTan* jest dana typu *real* stanowiąca arcus tangens danej reprezentowanej przez *Num*.

Przykładowo: *ArcTan*(0) = 0.0
ArcTan(1.0) = 0.7853981634

Funkcja *Cos*

Wywołanie: *Cos*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real* albo *integer*.

Rezultatem funkcji *Cos* jest dana typu *real* stanowiąca cosinus danej reprezentowanej przez *Num*.

Przykładowo: *Cos*(0) = 1.0
Cos(*Pi*) = −1.0

Funkcja *Exp*

Wywołanie: *Exp*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real* albo *integer*.

Rezultatem funkcji *Exp* jest dana typu *real* uzyskana z podniesienia do potęgi Num podstawy logarytmów naturalnych e = 2.7182818285

Przykładowo: *Exp*(0) = 1.0
Exp(−1.0) = 0.36787944117

Funkcja *Frac*

Wywołanie: *Frac*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real* albo *integer.*

Rezultatem funkcji *Frac* jest dana typu *real* stanowiąca część ułamkową danej reprezentowanej przez *Num.*

Przykładowo: *Frac*(3) = 0.0
Frac(*Pi*) = 0.1415926536

Funkcja *Int*

Wywołanie: *Int*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real* albo *integer.*

Rezultatem funkcji *Int* jest dana typu *real* stanowiąca część całkowitą danej reprezentowanej przez *Num.*

Przykładowo: *Int*(2) = 2.0
Int(*Pi*) = −3.0

Funkcja *Ln*

Wywołanie: *Ln*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real* albo *integer.*

Rezultatem funcji *Ln* jest dana typu *real* stanowiąca logarytm naturalny danej reprezentowanej przez *Num.*

Przykładowo: *Ln*(1) = 0.0
Ln(3.0) = 1.0986122887

Funkcja *Sin*

Wywołanie: *Sin*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real* albo *integer.*

Rezultatem funkcji *Sin* jest dana typu *real* stanowiąca sinus danej reprezentowanej przez *Num.*
Przykładowo: *Sin*(0) = 0.0
Sin(*Pi*/2) = 1.0

Funkcja *Sqr*

Wywołanie: *Sqr*(*Num*)

Przyjmuje się, że *Num* jęst wyrażeniem typu *real* albo *integer.*

Rezultatem funkcji *Sqr* jest dana takiego samego typu jak typ argumentu, stanowiąca kwadrat danej reprezentowanej przez *Num*.

Przykładowo: *Sqr*(2) = 4
Sqr(2.0) = 4.0

Funkcja *Sqrt*

Wywołanie: *Sqrt*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real* albo *integer*.

Rezultatem funkcji *Sqrt* jest dana typu *real*, stanowiąca pierwiastek kwadratowy danej reprezentowanej przez *Num*.

Przykładowo: *Sqrt*(4) = 2.0
Sqrt(4.0) = 2.0

Funkcje porządkowe

Funkcja *Odd*

Wywołanie: *Odd*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Rezultatem funkcji *Odd* jest dana typu *boolean*, wyrażająca wartość logiczną zdania „wyrażenie *Num* reprezentuje daną o wartości nieparzystej".

Przykładowo: *Odd*(3) = *true*
Odd(4) = *false*

Funkcja *Pred*

Wywołanie: *Pred*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem dowolnego typu porządkowego.

Rezultatem funkcji *Pred* jest dana takiego samego typu jak typ argumentu, stanowiąca poprzednik danej reprezentowanej przez *Num*, w jego typie porządkowym.

Przykładowo: *Pred*('*B*') = '*A*'
Pred(−20) = −21

Funkcja *Succ*

Wywołanie: *Succ*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem dowolnego typu porządkowego.

Rezultatem funkcji *Succ* jest dana takiego samego typu jak typ argumentu, stanowiąca następnik danej reprezentowanej przez *Num*, w jego typie porządkowym.

Przykładowo: *Succ*(*false*) = *true*
Succ(0) = 1

Funkcje do wykonywania konwersji

Funkcja *Chr*

Wywołanie: *Chr*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu integer.

Rezultatem funkcji *Chr* jest dana typu *char*, reprezentująca znak o kodzie przez *Num*.

Przykładowo: *Chr*(65) = '*A*'
Chr(27) = ^[

Funkcja *Ord*

Wywołanie: *Ord*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu dowolnego typu porządkowego.

Rezultatem funkcji *Ord* jest dana typu *integer*, określająca numer porządkowy danej reprezentowanej przez *Num*, w jego typie porządkowym.

Przykładowo: *Ord*(*true*) = 1
Ord(−2) = −2

Funkcja *Round*

Wywołanie: *Round*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real*.

Rezultatem funkcji *Round* jest dana typu *integer* o wartości najbliższej wartości danej reprezentowanej przez *Num*.

Przykładowo: *Round*(5.5) = 6
Round(−1.5) = −2

Funkcja *Trunc*

Wywołanie: *Trunc*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *real*.

Rezultatem funkcji *Trunc* jest dana typu *integer* stanowiąca część całkowitą danej reprezentowanej przez *Num*.

Przykładowo: *Trunc*(3.14) = 3
Trunc(−2.9) = −2

Funkcje pomocnicze

Funkcja *Hi*

Wywołanie: *Hi*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Rezultatem funkcji *Hi* jest dana typu *integer* o wartości bardziej znaczącego bajtu danej reprezentowanej przez *Num*.

Przykładowo: *Hi*(256) = 1
Hi(−1) = 255

Funkcja *KeyPressed*

Wywołanie: *KeyPressed*

Funkcja *KeyPressed* jest bezparametrowa.

Rezultatem funkcji *KeyPressed* jest dana typu *boolean* wyrażająca wartość logiczną zdania „w buforze wejściowym konsoli znajduje się nie wprowadzony jeszcze znak".

Funkcja *Lo*

Wywołanie: *Lo*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Rezultatem funkcji *Lo* jest dana typu *integer* o wartości mniej znaczącego bajtu danej reprezentowanej przez *Num*.

Przykładowo: *Lo*(256) = 0
Lo(−1) = 255

Funkcja *Random*

Wywołanie: *Random*
Random(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Rezultatem funkcji *Random* w wersji bezparametrowej jest dana typu *real* o wartości przypadkowej większej lub równej 0.0 i mniejszej niż 1.0. Rezultatem funkcji *Random* w wersji z parametrem jest dana typu *integer* o wartości przypadkowej większej lub równej 0 i mniejszej niż *Num*.

Funkcja *ParamCount*

Wywołanie: *ParamCount*

Funkcja *ParamCount* jest bezparametrowa.

Rezultatem funkcji *ParamCount* jest dana typu *integer* określająca liczbę parametrów przekazanych programowi w chwili jego wywołania. Zakłada się, że parametry zostały oddzielone spacjami lub znakami tabulacji.

Funkcja *ParamStr*

Wywołanie: *ParamStr*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Rezultatem funkcji *ParamStr* jest dana łańcuchowa reprezentująca ten parametr przekazany programowi w chwili jego wywołania, który ma numer *Num*. Zakłada się, że pierwszy parametr ma numer 1.

Funkcja *SizeOf*

Wywołanie: *SizeOf*(*Nam*)

Przyjmuje się, że *Nam* jest nazwą zmiennej albo nazwą typu.

Rezultatem funkcji *SizeOf* jest dana typu *integer* określająca liczbę bajtów pamięci przydzielonych zmiennej *Nam* albo liczbę bajtów pamięci niezbędnych do reprezentowania danej typu *Nam*.

Przykładowo: *SizeOf*(*boolean*) = 1
SizeOf(*integer*) = 2

Funkcja *Swap*

Wywołanie: *Swap*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Rezultatem funkcji *Swap* jest dana typu *integer*, której bardziej znaczący bajt ma wartość *Lo*(*Num*), a mniej znaczący bajt ma wartość *Hi*(*Num*).

Przykładowo: *Swap*(256) = 1
Swap(1) = 256

Funkcja *UpCase*

Wywołanie: *UpCase(Chr)*

Przyjmuje się, że *Chr* jest wyrażeniem typu *char*.

Rezultatem funkcji *UpCase* jest dana typu *char*, która jeśli *Chr* reprezentuje literę – jest literą dużą, a jeśli nie reprezentuje litery – jest taka jak dana reprezentowana przez *Chr*.

Przykładowo: *UpCase('a') = 'A'*
UpCase('.') = '.'

Przykład

```
program PrintArguments;
var
  Count : byte;
begin
  Write('Arguments are:');
  for Count := 1 to ParamCount do
    Write(' ',ParamStr(Count))
end.
```

- Jeśli program *PrintArguments* zostanie wywołany z argumentami

Jan Ewa Iza

to jego wykonanie spowoduje wyprowadzenie napisu

Arguments are: Jan Ewa Iza

- Jeśli zostanie wywołany bez argumentów, to jego wykonanie spowoduje wyprowadzenie napisu

Arguments are: □

17. Włączanie zbiorów

Jedną z ważniejszych dyrektyw kompilatora jest dyrektywa włączenia zbioru. Ma ona postać

{$*I name*}

gdzie *name* jest nazwą zbioru.

Zinterpretowanie takiej dyrektywy powoduje zastąpienie jej zawartością zbioru o nazwie *name*. Umożliwia to włączenie do programu uprzednio przygotowanych zestawów deklaracji albo bibliotek podprogramów w postaci źródłowej. Wymaga się jedynie, aby zbiór włączany nie zawierał dyrektyw włączania zbiorów.

Należy uwzględnić także to, że jeśli nazwa *name* nie zawiera czteroznakowego przyrostka, to nawias klamrowy kończący dyrektywę włączania zbioru powinien być od *name* oddzielony przynajmniej jedną spacją. W przeciwnym razie klamra zamykająca zostanie uznana za część nazwy zbioru.

Przykład
Jeśli przyjąć, że w zbiorze *Silnia.doc* znajduje się tekst

```
function Silnia(Num : byte) : integer;
begin
  if Num < 2 then Silnia := 1
             else Silnia := Num * Silnia(Num - 1)
end;
```

to skompilowanie programu

```
program Demo;
{$i Silnia.doc}
begin
  Writeln(Silnia(5))
end.
```

będzie miało taki sam skutek jak skompilowanie programu

```
program Demo;
function Silnia(Num : byte) : integer;
begin
  if Num < 2 then Silnia:= 1
             else Silnia:= Num * Silnia(Num - 1)
end;
begin
  Writeln(Silnia(5))
end.
```

□

18. Nakładkowanie podprogramów

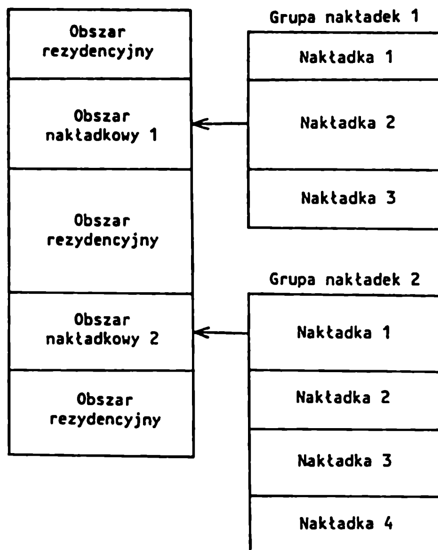
Istotnym ograniczeniem systemu Turbo Pascal jest założenie, że rozmiar kodu programu wynikowego oraz rozmiar jego danych nie przekracza 64K bajtów pamięci operacyjnej. Osłabienie skutków tego ograniczenia można uzyskać posługując się *nakładkowaniem*.

Nakładkowaniu mogą podlegać zarówno funkcje, jak i procedury. Wymaga się jedynie, aby podprogramy nakładkowane nie były wywoływane rekurencyjnie oraz aby nie były przedmiotem deklaracji zapowiadających. Nie stanowi to istotnego utrudnienia, ponieważ nic nie stoi na przeszkodzie zadeklarowania podprogramu nienakładkowanego wywołującego podprogram, który miał być np. wywołany rekurencyjnie.

Pewne trudności mogą powstać podczas wyszukiwania błędów w podprogramach nakładkowanych. Żeby nie wdawać się w opisy sposobów wykrywania takich błędów, wystarczy poprzestać na radzie, aby podprogramy zwykłe przekształcano w nakładkowane dopiero po ich uruchomieniu.

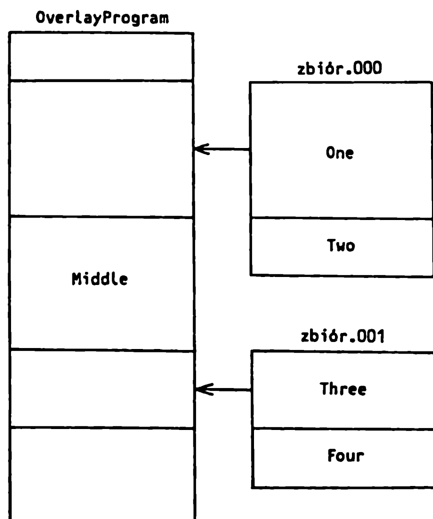
Zasadę nakładkowania przedstawiono na rys. 18.1. Polega ona na tym, że obszar pamięci operacyjnej przydzielony programowi wynikowemu zostaje podzielony na dwa rodzaje obszarów: obszary, w których zostanie ulokowana część rezydencyjna programu i obszary, do których będą sprowadzane nakładki.

Każda *nakładka* składa się z grupy podprogramów, której przypisano *obszar nakładkowy*. W każdej chwili wykonywania programu w obszarze nakładkowym może znajdować się co najwyżej jeden podprogram grupy. Rozmiar obszaru nakładkowego jest tak dobrany, że mieści się w nim każdy z podprogramów grupy. Oznacza to, że oszczędność pamięci operacyjnej wynikająca z posłużenia się grupą nakładkową wyraża się różnicą między rozmiarem grupy a rozmiarem największej nakładki w grupie.



Rys. 18.1 Zasada nakładkowania

Ceną, jaką płaci się za oszczędność pamięci operacyjnej, jest zakaz wzajemnego odwoływania się podprogramów grupy oraz konieczność wykonywania transmisji dyskowych podczas odwoływania się do tych podprogramów, które należą do grupy nakładkowej, ale w chwili wywołania nie znajdują się w pamięci operacyjnej.



Rys. 18.2 Struktura programu *Overlay Program*

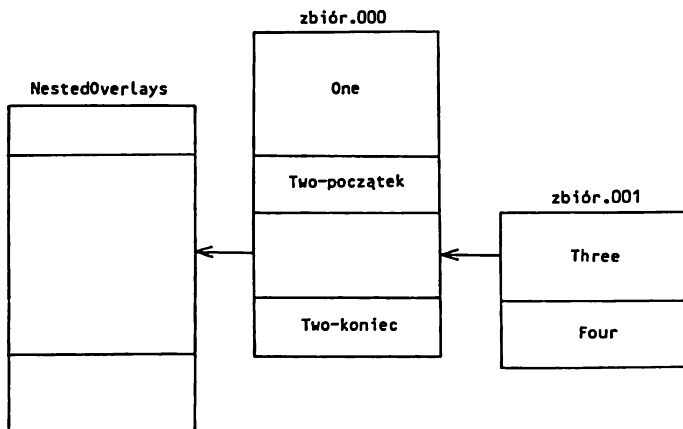
Zasady tworzenia grup nakładkowych są bardzo proste. W celu utworzenia grupy wystarczy poprzedzić słowem kluczowym **overlay** jedną lub więcej następujących po sobie deklaracji podprogramów. Wszystkie takie podprogramy będą wówczas stanowić grupę, zapamiętaną jako zbiór o nazwie programu głównego (dyrektywa M menu systemu) z trzycyfrowym przyrostkiem. Pierwsza grupa uzyska nazwę z przyrostkiem .000, druga z przyrostkiem .001 itd. Jeśli między podprogramami nakładkowanymi wystąpią podprogramy nienakładkowane, to podprogramy nakładkowane znajdą się w rozłącznych grupach. Zilustrowano to na rys. 18.2, na którym przedstawiono strukturę programu wynikowego powstałego z programu

```
program OverlayProgram;
overlay procedure One;
begin
  ...
end;
overlay procedure Two;
begin
  ...
end
procedure Middle;
begin
  ...
end;
overlay procedure Three;
begin
  ...
end;
overlay procedure Four;
begin
  ...
end;
begin
  ...
end.
```

Zasady tworzenia grup nakładkowych nie wykluczają możliwości tworzenia nakładek w obrębie podprogramów nakładkowanych. Przedstawiono to na rys. 18.3 dla programu wynikowego powstałego z programu

```
program NestedOverlays;
overlay procedure One;
begin
  ...
```

```
end;
overlay function Two(Num : byte) : byte;
  overlay function Three : real;
  begin
    ...
  end;
  overlay procedure Four;
  begin
    ...
  end;
begin {Two}
  ...
end;
begin {NestedOverlay}
  ...
end.
```



Rys. 18.3 Struktura programu *NestedOverlays*

19. Wybrane rozszerzenia implementacyjne

System Turbo Pascal, rozumiany jako połączenie języka programowania i zintegrowanego z nim edytora ekranowego, może być instalowany w środowisku różnych systemów operacyjnych, takich jak np. DOS i CP/M. W każdym z nich udostępnia on dodatkowe możliwości specyficzne dla obranego systemu i komputera. Do możliwości tych należą m. in.

- Jawne przydzielanie miejsca zmiennym programu.
- Dodatkowe funkcje i procedury systemowe.
- Bezpośrednie wywoływanie podprogramów systemowych.
- Programowanie w języku asemblera.

Mając na względzie ewentualne przenoszenie programów między systemami, zostaną tu omówione przykładowe rozszerzenia dla systemów DOS i CP/M. W celu umożliwienia selektywnego zapoznawania się z przytoczonym materiałem opisy te potraktowano rozłącznie.

System operacyjny DOS

Jawne przydzielanie miejsca zmiennym programu

Użycie w deklaracji pewnej zmiennej słowa kluczowego **absolute** umożliwia zarezerwowanie miejsca dla tej zmiennej w ustalonym obszarze pamięci operacyjnej, np. w miejscu przydzielonym innej zmiennej.

Deklaracja zawierająca słowo kluczowe **absolute** składa się z nazwy zmiennej deklarowanej, po której następuje dwukropek, określenie typu zmiennej, słowo kluczowe **absolute**, określenie miejsca, które ma zostać przydzielone zmiennej i średnik. Określenie miejsca może mieć postać nazwy zmiennej uprzednio zadeklarowanej, albo postać adresu bezwzględnego wyrażonego za pomocą oddzielonych dwukropkiem dwóch literałów typu *integer* albo odwołań do funkcji *Cseg*, *Dseg* albo *Sseg*. Para tych literałów lub odwołań ustala adres segmentu i przemieszczenie w segmencie tego adresu pamięci operacyjnej, pod którym zostanie przydzielone miejsce dla deklarowanej zmiennej.

Składnia

deklaracja-ze-słowem-absolute:
 nazwa-deklarowana : *oznaczenie-typu*
 absolute *określenie-miejsca*
nazwa-deklarowana:
 identyfikator
określenie-miejsca:
 określenie-miejsca-w-systemie-PC-DOS
 określenie-miejsca-w-systemie-CP/M-80
określenie-miejsca-w-systemie-PC-DOS:
 nazwa-zmiennej
 adres-segmentu : *przemieszczenie-w-segmencie*
adres-segmentu:
 literał
 nazwa-literału
 Cseg
przemieszczenie-w-segmencie:
 literał
 nazwa-literału
 Dseg
określenie-miejsca-w-systemie-CP/M-80:
 nazwa-zmiennej
 literał
 nazwa-literału
nazwa-zmiennej:
 identyfikator

Przykłady

```
var
  CharStr : string[6];
  StrLen : byte absolute CharStr;
  Para20 : array[0..15] of byte
            absolute 0020 : 0000;.
```

```
procedure Sub(var IntVar : integer);
var
  LSB : byte absolute IntVar;
begin
  if LSB = Lo(IntVar) then ...
  ...
end;
```

- Pierwszy bajt obszaru pamięci operacyjnej przydzielonego zmiennej *CharStr* pokrywa się z bajtem przydzielonym zmiennej *StrLen*. Odwołanie do zmiennej *StrLen* reprezentuje zatem taką samą daną jak odwołanie *Length*(*CharStr*) oraz *ord*(*CharStr*[0]).

- Zmiennej Para20 zostanie przydzielony obszar pamięci operacyjnej rozpoczynający się od bajtu o adresie 20 : 0, tj. od pierwszego bajtu paragrafu nr 20.

- Zmiennej *LSB* zostanie przydzielony ten sam bajt pamięci operacyjnej, który został przydzielony mniej znaczącemu bajtowi zmiennej, z którą skojarzono parametr *IntVar*. Z tego powodu relacja zawarta w procedurze *Sub* będzie zawsze prawdziwa. □

Dodatkowe funkcje i procedury systemowe

Funkcja *Addr*

Wywołanie: *Addr*(*Nam*)

Przyjmuje się, że *Nam* jest nazwą zmiennej.

Rezultatem funkcji *Addr* jest dana wskazująca określająca adres obszaru pamięci przydzielonego zmiennej albo podprogramowi *Nam*. Adres ten składa się z numeru segmentu i przemieszczenia w segmencie.

Przykładowo: *Addr*(*Mem*[2])

Funkcja *Ofs*

Wywołanie: *Ofs*(*Nam*)

Przyjmuje się, że Nam jest nazwą zmiennej.

Rezultatem funkcji *Ofs* jest dana typu *integer* określająca przemieszczenie w segmencie adresu obszaru przydzielonego zmiennej *Nam*. Nazwa Nam może być nazwą całościową albo nazwą częściową.

Przykładowo: *Ofs*(*ArrOfRec*[2,2].*LevOne*.*LevTwo*)

Funkcja *Seg*

Wywołanie: *Seg*(*Nam*)

Przyjmuje się, że *Nam* jest nazwą zmiennej.

Rezultatem funkcji *Seg* jest dana typu *integer* określająca numer segmentu, adresu obszaru przydzielonego zmiennej *Nam*. Nazwa *Nam* może być nazwą całościową albo nazwą częściową.

Przykładowo: *Seg*(*CharString*[3])

Funkcja *Cseg*

Wywołanie: *Cseg*

Funkcja *Cseg* jest bezparametrowa.

Rezultatem funkcji *Cseg* jest dana typu *integer* określająca numer segmentu, od którego rozpoczyna się segment *Code*.

Funkcja *Dseg*

Wywołanie: *Dseg*

Funkcja *Dseg* jest bezparametrowa.

Rezultatem funkcji *Dseg* jest dana typu *integer* określająca numer segmentu, od którego rozpoczyna się segment *Data*.

Funkcja *Sseg*

Wywołanie: *Sseg*

Funkcja *Sseg* jest bezparametrowa.

Rezultatem funkcji *Sseg* jest dana typu *integer* określająca numer segmentu, od którego rozpoczyna się segment *Stack*.

Procedura *ChDir*

Wywołanie: *ChDir*(*Str*)

Przyjmuje się, że *Str* jest wyrażeniem łańcuchowym.

Wykonanie procedury *ChDir* powoduje zmianę bieżącego katalogu na katalog określony przez daną reprezentowaną przez *Str*.

Przykładowo: *ChDir*('\JB\TURBO')

Procedura *MkDir*

Wywołanie: *MkDir(Str)*

Przyjmuje się, że *Str* jest wyrażeniem łańcuchowym.

Wykonanie procedury *MkDir* powoduje utworzenie (w bieżącym katalogu) podkatalogu o nazwie określonej przez daną reprezentowaną przez *Str*.

Przykładowo: *MkDir('GRAPHICS')*

Procedura *RmDir*

Wywołanie: *RmDir(Str)*

Przyjmuje się, że *Str* jest wyrażeniem łańcuchowym.

Wykonanie procedury *RmDir* powoduje usunięcie z bieżącego katalogu, podkatalogu o nazwie określonej przez daną reprezentowaną przez *Str*.

Przykładowo: *RmDir('GRAPHICS')*

Procedura *GetDir*

Wywołanie: *GetDir(Drive,Str)*

Przyjmuje się, że *Drive* jest wyrażeniem typu *integer*, a *Str* jest nazwą zmiennej łańcuchowej.

Wykonanie procedury *GetDir* powoduje przypisanie zmiennej *Str* danej łańcuchowej określającej nazwę bieżącego katalogu stacji o numerze określonym przez *Drive* (0 = A, 1 = B, itd.).

Przykładowo: *GetDir(2,StrVar[3])*

Procedura *OvrPath*

Wywołanie: *OvrPath(Path)*

Przyjmuje się, że *Path* jest wyrażeniem łańcuchowym.

Wykonanie procedury *OvrPath* powoduje potraktowanie łańcucha znaków reprezentowanego przez *Path* jako nazwy katalogu i przyjęcie, że nakładki programu nakładkowanego znajdują się w tym właśnie katalogu. W zapisie nazwy katalogu znak . (kropka) oznacza katalog bieżący.
Przykładowo: *OvrPath('\JB\GROUP')*
OvrPath('.')

Bezpośrednie wywoływanie podprogramów systemowych

Bezpośrednie wywołanie funkcji systemu operacyjnego zapewnia procedura *MsDos*, której argumentem jest zmienna typu

```
record
  AX,BX,CX,DX,
  BP,SI,DI,DS,ES,Flags : integer
end
```

albo typu

```
record
  case boolean of
    false: (AX,BX,CX,DX,
            BP,SI,DI,DS,Flags : integer);
    true:  (AL,AH,BL,BH,
            CL,CH,DL,DH : byte)
end
```

W chwili podjęcia wykonywania procedury *MsDos* poszczególne pola wymienionych rekordów muszą zawierać wartości spodziewane przez system w rejestrach procesora. Po zakończeniu wykonywania omawianej procedury zawartość rejestrów procesora zostanie przeniesiona do pól wymienionych rekordów.

Przykład

```
program GetDate;
type
  DateStr = string[10];
function Date : DateStr;
type
  Registers = record
                AX,BX,CX,DX,
                BP,SI,DI,DS,Flags : integer
              end;
var
  Regs : Registers;
  Month,Day : string[2];
  Year : string[4];
begin
  Regs.AX := Swap($2A);
  MsDos(Regs);
  with Regs do begin
    Str(CX,Year);
```

```
      Str(Lo(DX),Day);
      Str(Hi(DX),Month)
    end;
    Date := Month+'/'+Day+'/'+Year
  end;
  begin {GetDate}
    Writeln(Date)
  end.
```

• Przed wywołaniem procedury *MsDos* w górnej połowie rejestru *AX* zostanie umieszczony kod $2A stanowiący żądanie pod adresem systemu, aby w rejestrach procesora *CX* i *DX* została umieszczona bieżąca data: w rejestrze *CX* — rok, w górnej połowie rejestru *DX* — miesiac i w dolnej połowie tego rejestru — dzień. □

W analogiczny sposób jak procedura *MsDos* może być wywoływana procedura *Intr*. Jest to procedura dwuparametrowa, której pierwszy argument jest wyrażeniem określającym numer przerwania systemowego, a drugi jest nazwą zmiennej takiego typu jak dla procedury *MsDos* i o podobnym przeznaczeniu.

Przykład

```
program GetTime;
type
  TimeStr = string[8];
function Time : TimeStr;
type
  Registers = record
                AL,AH,BL,BH,
                CL,CH,DL,DH : byte
              end;
var
  Regs : Registers;
  Hour,Min,Sec : string[2];
begin
  AH := $2C;
  Intr($21,Regs);
  with Regs do begin
    Str(CH,Hour);
    Str(CL,Min);
    Str(DH,Sec)
  end;
  Time := Hour+' : '+Min+' : '+Sec
end;
```

begin {GetDate}
  Writeln(*Time*)
end.

• Przed wywołaniem procedury *Intr* w górnej połowie rejestru *AX* zostaje umieszczony kod $2*C* stanowiący żądanie pod adresem systemu, aby w rejestrach procesora *CX* i *DX* został umieszczony bieżący czas: w górnej połowie rejestru *CX* — liczba godzin, w dolnej — liczba minut, a w górnej połowie rejestru *DX* — liczba sekund. □

Programowanie w języku asemblera

Mimo iż jakość programów generowanych przez system Turbo Pascal jest bardzo wysoka, zachodzi niekiedy potrzeba dołączenia do programu napisanego w języku Turbo Pascal podprogramu napisanego w asemblerze, albo przynajmniej potrzeba posłużenia się wstawką asemblerową. Celowi temu służą konstrukcje językowe wykorzystujące słowa kluczowe **external** i **inline**.

Jeśli w miejscu bloku stanowiącego ciało procedury podprogramu występuje konstrukcja

external *literal-łańcuchowy*

to oznacza to, że podprogram jest napisany w języku asemblerowym i znajduje się w zbiorze o nazwie określonej przez literał. Wymaga się, aby wykonanie takiego podprogramu nie spowodowało zmiany zawartości rejestrów BP, CS, DS i SS.

Przykład

function *Factorial*(*Arg* : *byte*) : *integer*;
  external '*SILNIA.COM*';

• Część deklaracyjną i wykonawczą funkcji *Factorial* zastąpiono konstrukcją zawierającą słowo kluczowe **external**.
• Funkcja *Factorial* jest napisana w języku asemblerowym i znajduje się w zbiorze o nazwie *SILNIA.COM*. □

Poza przedstawionym tu sposobem deklarowania podprogramów napisanych w asemblerze, zapewniono w Turbo Pascalu możliwość umieszczania w jednym zbiorze więcej niż jednego podprogramu asemblerowego.

Deklaracje podprogramów należących do wspólnego zestawu powinny składać się z deklaracji pierwszego podprogramu zestawu, takiej jak omawiana wyżej, w celu dla określenia nazwy zbioru zawierającego zestaw oraz deklaracji składających się z nagłówka i konstrukcji

external *nazwa-podprogramu* [*przemieszczenie*]

W takich konstrukcjach *nazwa-podprogramu* jest nazwą pierwszego podprogramu zestawu, a *przemieszczenie* jest adresem względnym w tabeli skoków znajdującej się na początku kodu asemblerowego zestawu. Przyjmuje się, że zerowy element tabeli skoków jest związany z pierwszym podprogramem zestawu.

Przykład

> **procedure** *One*;
>   external *'CLUSTER.PRG'*;
> **function** *Two* : *byte*;
>   **external** *One*[3];
> **procedure** *Three*(*Count* : *byte*);
>   **external** *One*[6];

- Zestaw składa się z podprogramów *One*, *Two* i *Three*.
- Pierwszym podprogramem zestawu jest *One*.
- Podprogramy *One*, *Two* i *Three* znajdują się w zbiorze *CLUSTER.PRG*.
- Każdy element tabeli skoków zajmuje 3 bajty. □

Zdarza się często, że wstawki asemblerowe wymagane w programie napisanym w języku wysokiego poziomu są na tyle proste, że wygodne byłoby umieszczenie ich bezpośrednio między instrukcjami programu. Możliwość taką zapewnia w Turbo Pascalu specjalna instrukcja inline.

Instrukcja inline składa się ze słowa kluczowego **inline**, bezpośrednio po którym następuje ujęty w nawiasy okrągłe ciąg elementów kodu. Elementy kodu są oddzielone kreskami ukośnymi, a każdy z nich składa się z elementów danych oddzielonych od siebie znakami + (plus) albo — (minus).

Elementem danych jest literał typu *integer*, identyfikator zmiennej, identyfikator podprogramu oraz wyrażone za pomocą znaku * (gwiazdka) oznaczenie licznika instrukcji.

Przykład

> **inline**(24/$*FAFF*/*Source*—2/**Target*+3)

Każdy element danych generuje jeden bajt albo jedno słowo (dwa bajty) kodu. Każda wygenerowana dana wynika z wykonania działań na elementach danych. Przyjmuje się że identyfikator zmiennej albo podprogramu reprezentuje adres (ściślej przemieszczenie) tej zmiennej albo tego podprogramu. Analogicznie przyjmuje się, że oznaczenie licznika instrukcji reprezentuje adres tego najbliższego bajtu pamięci, w którym zostanie umieszczony wygenerowany kod.

Jeśli element kodu składa się wyłącznie z literałów i separatorów, a jego wartość mieści się w zakresie 0..255, to jest generowany jeden bajt. Jeśli wartość

elementu kodu nie mieści się we wspomnianym zakresie, jak również wtedy, gdy element kodu zawiera nazwy zmiennych lub podprogramów albo oznaczenia licznika instrukcji, to jest generowane jedno słowo. W słowie tym bajt mniej znaczący jest generowany przed bajtem bardziej znaczącym.

Przytoczone domniemania liczby generowanych bajtów kodu mogą zostać zmienione, jeśli przed elementem kodu zostanie umieszczony znak < (mniejsze) albo > (większe). W pierwszym przypadku zostanie wygenerowany tylko mniej znaczący z tych bajtów, w których jest reprezentowana wartość elementu kodu, a w drugim dwa bajty i to nawet wtedy, gdy drugi z nich (bardziej znaczący) reprezentuje daną o wartości 0.

Przykład

```
inline(>$12/<$3456)
```

• Pierwszy element kodu reprezentuje daną jednobajtową, ale wobec użycia znaku > (większe), generuje dwa bajty: $12 i $00.
• Drugi element kodu reprezentuje daną dwubajtową, ale wobec użycia znaku < (mniejsze), generuje jeden bajt: $56.
• Przytoczona instrukcja inline generuje trzy bajty kodu: $12, $00 i $56, w podanej kolejności.
• Gdyby rozpatrywaną instrukcję zmieniono do postaci

```
inline($12/$3456)
```

to generowałaby ona trzy bajty: $12, $56 i $34. □

Wymienione uprzednio przemieszczenia są wyznaczane w następujący sposób

• Adresy zmiennych zadeklarowanych w bloku programu są wyznaczane względem segmentu danych, wskazywanego przez rejestr *DS*.
• Adresy zmiennych zadeklarowanych w blokach podprogramów są wyznaczane względem segmentu stosu, wskazywanego przez rejestr *BP*.
• Adresy zmiennych inicjowanych są wyznaczane względem segmentu kodu, wskazywanego przez rejestr *CS*.

Przykład
(Przykład podano wg Turbo Pascal Reference Manual, v.3.0, rozdz. 20 — za zgodą Borland International)

```
type
  AnyString = string[255];
procedure ToUpper(var Str : AnyString);
begin
  inline(
    $c4/$be/Str/
    $26/$8a/$0d/
```

```
        $fe/$c1/
        $fe/$c9/
        $74/$13/
        $47/
        $26/$80/$3d/$61/
        $72/$f5/
        $26/$80/$3d/$7a
        $77/$ef/
        $26/$80/$2d/$20/
        $eb/$e9)
end;
```

- Wykonanie przytoczonej procedury powoduje zmianę małych liter zmiennej, z którą skojarzono parametr *Str*, na duże.
- Elementy kodu zawarte w instrukcji inline generują następujący podprogram asemblerowy

```
    LES DI,Str[BP]
    MOV CL,ES:[DI]
    INC CL
L1: DEC CL
    JZ L2
    INC DI
    CMP ES:BYTE PTR[DI],'a'
    JB L1
    CMP ES:BYTE PTR[DI],'z'
    JA L1
    SUB ES:BYTE PTR[DI],20H
    JMP SHORT L1
L2:
```
□

System operacyjny CP/M

Jawne przydzielanie miejsca zmiennym programu

Użycie w deklaracji pewnej zmiennej słowa kluczowego **absolute** umożliwia zarezerwowanie miejsca dla tej zmiennej w miejscu przydzielonym innej zmiennej, albo w ustalonym obszarze pamięci operacyjnej.

Deklaracja zawierająca słowo kluczowe **absolute** składa się z nazwy zmiennej deklarowanej, po której następuje dwukropek, określenie typu zmiennej, słowo

kluczowe **absolute**, określenie miejsca, które ma zostać przydzielone zmiennej i średnik. Określenie miejsca może mieć postać nazwy uprzednio zadeklarowanej albo postać adresu bezwzględnego wyrażonego za pomocą literału typu *integer*. Wartość tego literału ustala adres w pamięci operacyjnej, pod którym zostanie przydzielone miejsce dla deklarowanej zmiennej.

Składnia

deklaracja-ze-słowem-absolute:
nazwa-deklarowana : *oznaczenie-typu* **absolute** *określenie-miejsca*
nazwa-deklarowana:
identyfikator
określenie-miejsca:
nazwa-zmiennej:
literał
nazwa-zmiennej:
identyfikator

Przykłady

```
var
  CharStr : string[6];
  StrLen : byte absolute CharStr;
  Buffer : array[0..15] of byte
           absolute $2000;
  function Fun(var IntVar : integer) : byte;
  var
    LSB : byte absolute IntVar;
  begin
    if LSB = Lo(IntVar) then ...
    ...
  end;
```

• Pierwszy bajt obszaru pamięci operacyjnej, przydzielonego zmiennej *CharStr*, pokrywa się z bajtem przydzielonym zmiennej *StrLen*. Odwołanie do *StrLen* reprezentuje zatem taką samą daną jak odwołanie *Length*(*CharStr*) oraz *ord*(*CharStr*[0]).

• Zmiennej *Buffer* zostanie przydzielony obszar pamięci operacyjnej rozpoczynający się od bajtu o adresie $2000.

• Zmiennej *LSB* zostanie przydzielony ten sam bajt pamięci operacyjnej, który został przydzielony mniej znaczącemu bajtowi zmiennej, z którą skojarzono parametr *IntVar*. Z tego powodu relacja zawarta w funkcji *Fun* będzie zawsze prawdziwa. □

Dodatkowe funkcje i procedury systemowe

Funkcja *Addr*

Wywołanie: *Addr*(*Nam*)

Przyjmuje się, że *Nam* jest nazwą zmiennej albo podprogramu.

Rezultatem funkcji *Addr* jest dana wskazująca, określająca adres obszaru pamięci przydzielonego zmiennej albo podprogramowi *Nam*.

Przykładowo: *Addr*(*Mem*[2])

Procedura *OvrDrive*

Wywołanie: *OvrDrive*(*Drv*)

Przyjmuje się, że *Drv* jest wyrażeniem typu *integer*.

Wykonanie procedury *OvrDrive* powoduje, że w programie nakładkowanym, nakładki będą poszukiwane w stacji dyskowej określonej przez *Drv* (0 = stacja domniemana, 1 = A, 2 = B itd.)

Przykładowo: *OvrDrive*(2)

Bezpośrednie wywoływanie podprogramów systemowych

Bezpośrednie wywoływanie podprogramów systemowych zapewniają procedury *Bdos* i *Bios* oraz funkcje *Bdos*, *BdosHL*, *Bios* i *BiosHL*.

Procedura *Bdos*

Wywołanie: *Bdos*(*Fun*,*Par*)
Bdos(*Fun*)

Przyjmuje się, że *Fun* oraz nie wymagane *Par* są wyrażeniami typu *integer*.

Wykonanie procedury *Bdos* powoduje załadowanie do rejestru *C* procesora mniej znaczącego bajtu danej reprezentowanej przez *Fun*, a następnie przejście do wykonywania programu począwszy od adresu 5 w celu zrealizowania funkcji systemu BDOS określonej przez *Fun*. Jeśli procedura jest wywoływana z dwoma argumentami, to przed wykonaniem wspomnianego skoku następuje umieszczenie danej reprezentowanej przez *Par* w rejestrze *DE*.

Funkcja *Bdos*

Wywołanie: *Bdos(Fun,Par)*
Bdos(Fun)

Przyjmuje się, że *Fun* oraz nie wymagane *Par* są wyrażeniami typu *integer*.

Skutki wywołania funkcji *Bdos* są takie same jak skutki wywołania procedury *Bdos* z takimi samymi argumentami. Rezultatem funkcji jest natomiast dana typu *byte* pozostawiana przez *Bdos* w rejestrze *A* procesora.

Funkcja *BdosHL*

Wywołanie: *BdosHL(Fun,Par)*
BdosHL(Fun)

Przyjmuje się, że *Fun* oraz nie wymagane *Par* są wyrażeniami typu *integer*.

Skutki wywołania funkcji *BdosHL* są takie same jak skutki wywołania funkcji *Bdos* z takimi samymi argumentami, z tą różnicą, że rezultatem funkcji jest dana typu *integer* pozostawiana przez BDOS w rejestrach *HL* procesora.

Procedura *Bios*

Wywołanie: *Bios(Fun,Par)*
Bios(Fun)

Przyjmuje się, że *Fun* oraz nie wymagane *Par* są wyrażeniami typu *integer*.

Wykonanie procedury *Bios* powoduje wykonanie tej funkcji podsystemu BIOS, która ma numer *Fun*. Jeśli posłużono się argumentem *Par*, to przed wykonaniem tej czynności w rejestrach *BC* procesora jest umieszczana dana reprezentowana przez *Par*.

Funkcja *Bios*

Wywołanie: *Bios(Fun,Par)*
Bios(Fun)

Przyjmuje się, że *Fun* oraz nie wymagane *Par* są wyrażeniami typu *integer*.

Skutki wywołania funkcji *Bios* są takie same jak skutki wywołania procedury *Bios* z takimi samymi argumentami. Rezultatem funkcji jest natomiast dana typu *byte* pozostawiana przez BIOS w rejestrze *A* procesora.

Funkcja *BiosHL*

Wywołanie: *BiosHL(Fun,Par)*

Przyjmuje się, że *Fun* oraz nie wymagane *Par* są wyrażeniami typu *integer*.

Skutki wywołania funkcji *BiosHL* są takie same jak skutki wywołania funkcji *Bios* z takimi samymi argumentami, z tą różnicą, że rezultatem funkcji jest dana typu *integer* pozostawiana przez BIOS w rejestrach *HL* procesora.

Programowanie w języku asemblera

Niekiedy zachodzi potrzeba dołączenia do programu napisanego w języku Turbo Pascal podprogramu napisanego w asemblerze, albo przynajmniej potrzeba posłużenia się wstawką asemblerową. Celowi temu służą konstrukcje językowe wykorzystujące słowa kluczowe **external** i **inline**.

Jeśli zamiast bloku stanowiącego ciało podprogramu występuje konstrukcja

external *literał-typu-integer*

to oznacza to, że podprogram jest napisany w języku asemblerowym i znajduje się w pamięci operacyjnej pod adresem określonym przez literał. Wymaga się, aby wykonanie takiego podprogramu nie spowodowało zmiany rejestru *SP* (rozmiaru stosu).

Przykład

function *Factorial*(*Arg* : *byte*) : *integer*;
  external $6000;

- Część deklaracyjną i wykonawczą funkcji *Factorial* zastąpiono konstrukcją zawierającą słowo kluczowe **external**.
- Funkcja *Factorial* jest napisana w języku asemblerowym, a jej kod znajduje się w pamięci operacyjnej począwszy od adresu $6000. □

O ile konstrukcja external jedynie wskazuje miejsce, w którym znajduje się podprogram asemblerowy, o tyle instrukcja inline umożliwia wygenerowanie kodu asemblerowego.

Instrukcja inline składa się ze słowa kluczowego **inline**, bezpośrednio po którym następuje ujęty w nawiasy okrągłe ciąg elementów kodu. Elementy kodu są oddzielone kreskami ukośnymi, a każdy z nich składa się z elementów danych oddzielonych od siebie znakami + (plus) albo — (minus).

Elementem danych jest literał typu *integer*, identyfikator zmiennej, identyfikator podprogramu oraz wyrażone za pomocą znaku * (gwiazdka) oznaczenie licznika instrukcji.

Przykład

inline(20/$40/*Fun* – 2/* + 3) □

Każdy element danych generuje jeden bajt albo jedno słowo (dwa bajty) kodu. Każda wygenerowana dana wynika z wykonania działań na elementach

danych. Przyjmuje się, że identyfikator zmiennej albo podprogramu reprezentuje adres tej zmiennej albo podprogramu. Analogicznie przyjmuje się, że oznaczenie licznika instrukcji reprezentuje adres tego najbliższego bajtu pamięci, w którym zostanie umieszczony wygenerowany kod.

Jeśli element kodu składa się wyłącznie z literałów i separatorów, a jego wartość mieści się w zakresie 0..255, to zostanie wygenerowany jeden bajt kodu. Jeśli wartość wykracza poza ten zakres, jak również wtedy, gdy element kodu zawiera nazwy zmiennych lub podprogramów albo odwołania do licznika instrukcji, jest generowane jedno słowo. W słowie tym bajt mniej znaczący jest generowany przed bajtem bardziej znaczącym.

Przytoczone domniemania liczby generowanych bajtów kodu mogą zostać zmienione, jeśli przed elementem kodu zostanie umieszczony znak < (mniejsze) albo > (większe). W pierwszym przypadku zostanie wygenerowany tylko mniej znaczący z tych bajtów, w których jest reprezentowana wartość elementu kodu, a w drugim dwa bajty i to nawet wtedy, gdy drugi z nich (bardziej znaczący) reprezentuje daną o wartości 0.

Przykład

```
inline (>$12<3456)
```

- Pierwszy element kodu reprezentuje daną jednobajtową, ale wobec użycia znaku > (większe) generuje dwa bajty: $12 i $00.
- Drugi element kodu reprezentuje daną dwubajtową, ale wobec użycia znaku < (mniejsze) generuje jeden bajt: $56.
- Przytoczona instrukcja inline generuje trzy bajty kodu: $12, $00, $56, w podanej kolejności.
- Gdyby rozpatrywaną instrukcję zmieniono do postaci

```
inline($12/$3456)
```

to generowałaby ona trzy bajty: $12, $56 i $34. □

Przykład
(wg Turbo Pascal Reference Manual, wersja 3.0, rozdz. 22 — za zgodą Borland International)

```
type
  AnyString = string[255];
procedure ToUpper(var Str : AnyString);
begin
  inline(
    $2a/Str/
    $46/
    $04/
```

```
        $05/
        $ca/*+20/
        $23/
        $7e/
        $fe/$61/
        $da/*-9/
        $fe/$7b/
        $d2/*-14/
        $d6/$20/
        $77/
        $c3/*-20)
end;
```

• Wykonanie przytoczonej procedury powoduje zamianę małych liter zmiennej łańcuchowej, z którą skojarzono parametr *Str*, na duże.
• Elementy kodu zawarte w instrukcji inline generują następujący podprogram asemblerowy:

```
    LD   HL,(Str)
    LD   B,(HL)
    INC  B
L1: DEC  B
    JP   Z,L2
    INC  HL
    LD   A,(HL)
    CP   'a'
    JP   C,L1
    CP   'z'+1
    JP   NC,L1
    SUB  20H
    LD   (HL),A
    JP   L1
L2:
```
□

Reprezentowanie danych

Sposób reprezentowania danych w pamięci operacyjnej jest w systemach DOS i CP/M niemal taki sam, gdyż mało istotna różnica dotyczy jedynie danych wskazujących. Znacznie ułatwia to przenoszenie programów między tymi systemami.

Dane porządkowe

Dane typu porządkowego są reprezentowane w jednym albo w dwóch bajtach pamięci operacyjnej.

W jednym bajcie są reprezentowane dane typu *char*, dane typu wyliczeniowego związanego ze zbiorem liczącym nie więcej niż 256 elementów oraz dane typu okrojonego *min..max*, takiego, że zarówno *ord(min)* jak *ord(max)* należy do przedziału 0..255. W szczególności w jednym bajcie są więc reprezentowane dane typu *boolean* i *byte*.

W dwóch bajtach są reprezentowane dane typu *integer*, dane typu okrojonego, które nie mogą być reprezentowane w jednym bajcie oraz dane typu wyliczeniowego związanego ze zbiorem o więcej niż 256 elementach.

W przypadku danych zajmujących dwa bajty mniej znaczącym bajtem danych jest pierwszy z nich.

Przykład

```
type
   Color = (Red,Green,Blue,Yellow,Orange);
   Hue = Green..Yellow;
   Selector = (enum,card,bool);
   union = record
               case Selector of
                  enum: (Rng : Hue);
                  card: (Int : integer);
                  bool: (Log : boolean)
               end;
const
   HueVar : union = (Rng : Green);
begin
   with HueVar do
      Write(Int,Log :5)
end.
```

- Ponieważ *ord(Green)* ⩽ 255 oraz *ord(Yellow)* ⩽ 255, pole *Rng typu Hue* jest reprezentowane w jednym bajcie.
- Mniej znaczący bajt pola *Int* pokrywa się z bajtem pola *Rng* i bajtem pola *Log*.
- Wkonanie instrukcji *Write* powoduje wyprowadzenie napisu

    1 TRUE □

Dane rzeczywiste

Dane typu *real* są reprezentowane w 6 bajtach pamięci operacyjnej.

Dane te składają się z jednobajtowego wykładnika i pięciobajtowej mantysy. Pod najniższym adresem pamięci występuje wykładnik, a za nim mantysa, w kolejności od bajtu najmniej znaczącego do najbardziej znaczącego. Wartość wykładnika jest reprezentowana z przesunięciem $80, a mantysa jest znormalizowana i pozbawiona najbardziej znaczącego bitu. Bit ten jest wykorzystany do reprezentowania jej znaku. Dana rzeczywista typu *real* o wartości 0.0 jest reprezentowana przez wykładnik składający się z samych bitów 0.

Przykład

```
type
  union = record
            case boolean of
             false: (Float : real);
             true: (Arr : array[0..5] of byte)
           end;
const
  RealVar : union = (Arr : ($83,0,0,0,0,$80));
begin
  with RealVar do
    Write(Float)
end.
```

• Pierwszy bajt zmiennej *RealVar* jest daną bajtową o wartości $83, a więc reprezentuje wykładnik 3.
• Bajt o wartości $80 jest najbardziej znaczącym bajtem mantysy. Reprezentuje on znak – (minus) mantysy oraz mantysę

10000000 00000000 00000000 00000000 00000000

Najbardziej znaczący bit tego bajtu reprezentuje znak mantysy, tu – (minus).
• Ponieważ mantysa ma wartość −0.5, a wykładnik ma wartość 3, dana początkowa przypisana zmiennej *RealVar* ma wartość $-0.5*2^3 = -4.0$
• Wykonanie instrukcji *Write* powoduje wyprowadzenie liczby −4.0. □

Dane łańcuchowe

Dane typu **string**[*n*] są reprezentowane w $n+1$ bajtach pamięci. Pierwszy bajt danej łańcuchowej określa liczbę znaków tej danej.

Przykład

```
const
  StrVar : string[5] = 'Janek';
procedure CutOff(var Par);
var
  Len : byte absolute Par;
begin
  Len := Len-2
end;
begin
  Writeln(StrVar);
  CutOff(StrVar);
  Write(StrVar)
end.
```

- Operacja na zmiennej *Len* jest w istocie operacją na najbardziej znaczącym bajcie zmiennej *StrVar*.
- Wykonanie przytoczonego programu powoduje wyprowadzenie napisu

Janek
Jan □

Dane mnogościowe

Dane typu mnogościowego, bazowane na typie porządkowym o najmniejszym elemencie *min* i największym elemencie *max* zajmują

ord(*max*) **div** 8 − *ord*(*min*) **div** 8 + 1

bajtów pamięci.

Dane mnogościowe są reprezentowane w taki sposób, jakby bazowy typ porządkowy składał się z 256 elementów. Ponieważ wymagałoby to zarezerwowania dla danej 32 bajtów pamięci, z których wiele nie byłoby w większości przypadków używanych, z tych 32 bajtów odrzuca się te bajty skrajne, których wartość nigdy nie ulega zmianie.

W szczególności, dana typu **set of** 10..16 nie jest reprezentowana w postaci 32-bajtowej (literami x oznaczono aktywne pozycje danej)

00000000 ... 00000000 0000000x xxxxxx00 00000000

lecz w postaci 2-bajtowej

0000000x xxxxxx00

w kolejności od bajtu najmniej znaczącego do najbardziej znaczącego.

Przykład

```
type
  union = record
            case boolean of
              false : (aSet : set of 10..16);
              true : (anInt : integer)
          end;
const
  SetVar : union = (aSet : ([16]));
begin
  with SetVar do
    Write(anInt)
end.
```

• Pole *aSet* zajmuje 2 bajty i jest reprezentowane w pamięci operacyjnej w postaci 00000000 00000001.

• Wykonanie instrukcji *Write* powoduje wyprowadzenie liczby 256. □

Dane wskazujące

Dane wskazujące są w systemach DOS i CP/M reprezentowane odmiennie. W systemie DOS dana wskazująca jest reprezentowana w 4, a w systemie CP/M w 2 bajtach pamięci każda.

W pierwszym przypadku bardziej znacząca połowa reprezentacji danej zawiera numer segmentu, a mniej znacząca przemieszczenie w segmencie. Części te przechowywane są tak jak dane typu *integer*.

W drugim przypadku dana wskazująca jest reprezentowana tak jak dana typu *integer* i określa adres pamięci operacyjnej.

Dana reprezentowana przez **nil** składa się z samych bitów 0.

Przykład

```
type
  union = record
            case boolean of
              false: (Ref : ^byte);
              true: (Int : integer)
          end;
const
  PtrVar : union = (Int : 0);
begin
  Write(PtrVar = nil)
end.
```

- W systemie DOS zmienna *PtrVar* zajmuje 4 bajty, a w systemie CP/M zajmuje 2 bajty.
- Wykonanie instrukcji *Write* powoduje wyprowadzenie napisu TRUE. □

Dane tablicowe

Elementy tablic są rozmieszczone w pamięci operacyjnej wierszami.

Przykład

```
type
  union = record
            case boolean of
              false : (Arr : array[boolean, boolean] of byte
              true: (Vec : array[1..4] of byte)
          end;
const
  ArrVar : union = ((11, 12),((21, 22));
var
  Index : 1..4;
begin
  with ArrVar do
    for Index:= 1 to 4 do
      Write(Vec[Index] :3)
end.
```

- Wykonanie instrukcji *Write* powoduje wyprowadzenie liczb 11, 12, 21, 22. □

Dane rekordowe

Pola rekordów są rozmieszczone w pamięci operacyjnej w kolejności ich wystąpienia w deklaracji. Jeśli rekord nie zawiera wariantów, to jego rozmiar jest równy sumie rozmiarów pól. W przeciwnym razie rozmiar rekordu jest równy sumie rozmiaru jego części stałej i rozmiaru tego z wariantów, który wymaga najwięcej miejsca. Zarówno rekordy bez wariantów, jak i rekordy z wariantami są stałego rozmiaru. Informacja ta jest istotna z punktu widzenia operacji wejścia/wyjścia.

Przykład

```
var
  RecVar : record
             case Selector : integer of
               6: (Arr : array[1..3,1..2] of byte);
               8: (aSet : set of 'A'..'z')
           end;
```

```
begin
  with RecVar do
    Write (SizeOf(RecVar),
           SizeOf(Arr) :2,
           SizeOf(aSet) :2)
end.
```

- Część stała rekordu *RecVar* ma rozmiar 2 bajty.
- Pierwszy wariant tego rekordu ma rozmiar 6 bajtów.
- Drugi wariant ma rozmiar 8 bajtów.
- Rekord ma rozmiar 10 bajtów.
- Wykonanie instrukcji *Write* powoduje wyprowadzenie liczb 10, 6 i 8. □

20. Podstawy grafiki dla IBM PC

Chociaż tekst tego rozdziału dotyczy wyłącznie mikrokomputera IBM PC, przedstawione w nim środki programowe stanowią wzór dla implementacji na innych mikrokomputerach. Potwierdzeniem tej tezy jest pojawienie się na rynku analogicznych bibliotek podprogramów dla mikrokomputerów Amstrad 6128.

Monitor ekranowy

Współpraca z monitorem ekranowym może odbywać się w jednym z 4 trybów tekstowych albo w jednym z 3 trybów graficznych.

W *trybie tekstowym* ekran monitora jest podzielony na 25 wierszy, a w każdym wierszu znajdują się pozycje dla 40 albo 80 znaków. Na pozycjach tych mogą być wyświetlane znaki rozszerzonego kodu ASCII.

W *trybie graficznym* ekran monitora stanowi raster składający się z 200 linii. W każdej linii można pobudzić do świecenia 320 albo 640 punktów.

Zarówno w trybach tekstowych, jak i graficznych zapewniono środki do wyświetlania w kolorach.

Tryby tekstowe

Przełączenie procesora do trybu tekstowego wymaga wykonania procedury *TextMode*.

Procedura *TextMode*

Wywołanie: *TextMode(BW40)*
TextMode(C40)
TextMode(BW80)
TextMode(C80)
TextMode

Nazwy *BW40*, *C40*, *BW80* i *C80* są predefiniowanymi nazwami literałów typu *integer*, o wartościach odpowiednio 0, 1, 2 i 3.

Wykonanie procedury *TextMode* z argumentem *BW40* powoduje ustanowienie trybu graficznego znakowego z wyświetlaniem czarno-białym (Black and White) do 40 znaków w wierszu.

Wykonanie procedury *TextMode* z argumentem *C40* powoduje ustanowienie trybu znakowego z wyświetlaniem kolorowym (Color) do 40 znaków w wierszu.

Wykonanie procedury *TextMode* z argumentem *BW80* powoduje ustanowienie trybu tekstowego z wyświetlaniem czarno-białym do 80 znaków w wierszu.

Wykonanie procedury *TextMode* z argumentem *C80* powoduje ustanowienie trybu znakowego z wyświetlaniem kolorowym do 80 znaków w wierszu.

Wykonanie procedury *TextMode* bez argumentu powoduje ustanowienie ostatnio włączonego trybu tekstowego.

W każdym przypadku ustanowienie trybu tekstowego powoduje wyczyszczenie ekranu, tj. zapełnienie go spacjami, a ponadto przemieszczenie kursora do lewego-górnego narożnika ekranu.

Przykład

```
program Ewa;
begin
  TextMode;
  Write ('Ewa');
  TextMode (BW40)
end.
```

- Bezpośrednio po aktywowaniu programu, ale przed wykonaniem procedury *TextMode*, ekran może zawierać znaki różne od spacji.
- Wykonanie procedury *TextMode* bez argumentu powoduje wyczyszczenie ekranu i ustanowienie trybu tekstowego domniemanego C80.
- Po wyczyszczeniu ekranu kursor znajduje się w jego lewym-górnym narożniku.

• Wykonanie instrukcji *Write* powoduje wyprowadzenie w pierwszym wierszu ekranu napisu *Ewa.*
• Wykonanie procedury *TextMode* z argumentem *BW40* powoduje ustanowienie trybu tekstowego *BW40.*
• Po zakończeniu wykonywania programu wyprowadzanie na ekran pozostaje w trybie *BW40.* □

W trybach tekstowych z kolorem każdy znak może być wyświetlany w jednym z 16 kolorów na dowolnym tle wybranym z 8 kolorów. Na kolory można powoływać się za pomocą literałów typu *integer* albo za pomocą mnemonicznych nazw tych literałów, zgodnie z zestawieniem w tabl. 20.1. Znaki mogą być

Tablica 20.1. Mnemoniczne nazwy literałów oznaczających kolory

Literał	Nazwa
0	*Black* (czarny)
1	*Blue* (niebieski)
2	Green (zielony)
3	*Cyan* (turkusowy)
4	*Red* (czerwony)
5	*Magenta* (karmazynowy)
6	*Brown* (brązowy)
7	*LightGray* (jasnoszary)
8	*DarkGray* (ciemnoszary)
9	*LightBlue* (jasnoniebieski)
10	*LightGreen* (jasnozielony)
11	*LightCyan* (jasnoturkusowy)
12	*LightRed* (jasnoczerwony)
13	*LightMagenta* (jasnokarmazynowy)
14	*Yellow* (żółty)
15	*White* (biały)

wyświetlane zarówno w kolorach ciemnych, jak i jasnych. Tło mogą stanowić jedynie kolory ciemne. Należy nadmienić, że niektóre monitory kolorowe nie umożliwiają odróżnienia kolorów ciemnych od jasnych. W takim przypadku kolory jasne są wyświetlane tak jak ich ciemne odpowiedniki.

Jeśli do liczby określającej kolor zostanie dodane 16, to wyświetlany znak będzie migotać. W programie można to wyrazić, posługując się predefiniowaną nazwą literału *Blink* o wartości 16.

Określenie koloru wyświetlanych znaków odbywa się za pomocą procedury *TextColor*, a określenie koloru tła za pomocą procedury *TextBackground*. Dwie dodatkowe funkcje *WhereX* i *WhereY* umożliwiają określenie bieżącej pozycji kursora.

Procedura *TextColor*

Wywołanie: *TextColor(Hue)*

Przyjmuje się, że *Hue* jest wyrażeniem typu *integer* o wartości z przedziału 0..31.

Wykonanie procedury *TextColor* powoduje określenie koloru wyprowadzanych znaków, zgodnie z odwzorowaniem przytoczonym w tablicy 20.1. Jeśli dana reprezentowana przez *Hue* ma wartość większą niż 15, to argument jest traktowany tak, jakby miał wartość *Hue*−15, a wyświetlane znaki są pobudzane do migotania.

Przykład

```
program EwaIzaJan;
begin
  TextMode(C40);
  Write('Ewa');
  TextColor(Red);
  Write('Iza');
  TextColor(Green + Blink);
  Write('Jan');
  TextMode(BW80)
end.
```

- Napis *Ewa* zostanie wyprowadzony w kolorze domniemanym — żółtym i na tle domniemanym — czarnym.
- Napis *Iza* zostanie wyprowadzony na tle domniemanym i w kolorze czerwonym.
- Napis *Jan* zostanie wyprowadzony na tle domniemanym i w kolorze zielonym. Litery *Jan* będą migotać.
- Przed zakończeniem wykonywania programu zostanie przywrócony tryb tekstowy *BW80*. □

Procedura *TextBackground*

Wywołanie: *TextBackground(Hue)*

Przyjmuje się, że *Hue* jest wyrażeniem typu *integer* reprezentującym daną o wartości z przedziału 0..7.

Wykonanie procedury *TextBackground* powoduje określenie koloru tła wyświetlanych znaków.

Przykład

```
program Jan;
begin
  TextMode(C40)
  TextBackground(Green);
  TextColor(Brown);
  Write('Jan')
end.
```

• Napis *Jan* zostanie wyprowadzony w kolorze brązowym na zielonym tle. □

Funkcja *WhereX*

Wywołanie: *WhereX*

Funkcja *WhereX* jest bezparametrowa.

Rezultatem funkcji *WhereX* jest dana typu *integer*, określająca numer kolumny zawierającej znak wyróżniony przez kursor.

Przykład

```
program Column;
begin
  GotoXY(20,30);
  Write(WhereX)
end.
```

• Wykonanie instrukcji *Write* powoduje wyprowadzenie liczby 30. □

Funkcja *WhereY*

Wywołanie: *WhereY*

Funkcja *WhereY* jest bezparametrowa.

Rezultatem funkcji *WhereY* jest dana typu *integer*, określająca numer wiersza zawierającego znak wyróżniony przez kursor.

Przykład

```
program Default;
begin
  TextMode;
  Write(WhereX,WhereY :2)
end.
```

• Wykonanie instrukcji Write powoduje wyprowadzenie liczb 1, 1. □

Tryby graficzne

W języku Turbo Pascal zapewniono możliwość posługiwania się trzema trybami graficznymi ustanawianymi za pomocą wywołań procedur *GraphColorMode*, *GraphMode* i *HiRes*. Wywołanie każdej z tych procedur powoduje wyczyszczenie ekranu.

Przed zakończeniem wykonywania programu, w którym posługiwano się trybem graficznym, należy zapewnić ustanowienie jednego z trybów tekstowych.

Wyświetlanie graficzne może być kolorowe albo czarno-białe. Tło do wyświetlania punktów, nazywanych również *pikselami*, może być czarne albo kolorowe. Kolor tła może zostać określony za pomocą procedury *GraphBackground*.

Kolory pikseli mogą być wybierane z palety barw określanej za pomocą procedury *Palette*. Na monitorze kolorowym są dostępne 4 palety wyszczególnione w tabl. 20.2. Każda z palet umożliwia wybranie jednego z 4 kolorów.

Tablica 20.2. Palety barw dla monitora kolorowego

Numer koloru	0	1	2	3
Paleta 0	tło	zielony	czerwony	brązowy
Paleta 1	tło	turkusowy	karmazynowy	jasnoszary
Paleta 2	tło	jasnozielony	jasnoczerwony	żółty
Paleta 3	tło	jasnoturkusowy	jasnokarmazynowy	biały

Tablica 20.3. Palety barw dla monitora RGB

Numer koloru	0	1	2	3
Paleta 0	tło	niebieski	czerwony	jasnoszary
Paleta 1	tło	jasnoniebieski	jasnoczerwony	biały

Jeśli wyświetlanie odbywa się w trybie graficznym *GraphMode*, to jest ono czarno-białe. Tym niemniej — na monitorze RGB — istnieje możliwość wyboru barwy z 2 palet wyszczególnionych w tabl. 20.3. Jeśli wyświetlanie odbywa się w trybie graficznym wysokiej rozdzielczości, ustanawianym za pomocą procedury *HiRes*, to kolor wykresu może być określony za pomocą procedury *HiResColor*. Jeśli procedura ta nie zostanie wywołana, to wyświetlanie będzie odbywać się w domniemanym kolorze białym.

Ustanowienie trybu graficznego nie wyklucza możliwości wyprowadzania na ekran tekstu. Znaki takiego tekstu, wyprowadzane za pomocą procedury *Write*,

są umieszczane na takich samych pozycjach jak w trybach tekstowych. Jeśli tryb graficzny dopuszcza kolor, to znaki są wyświetlane w kolorze nr 3 bieżącej palety, na tle o kolorze nr 0. W trybie wysokiej rozdzielczości znaki są wyświetlane w kolorze określonym za pomocą procedury *HiResColor* albo w kolorze domniemanym – białym.

Podstawowymi obiektami graficznymi, które można wyświetlać za pomocą procedur graficznych, są punkt i odcinek. Wyświetlanie punktu jest realizowane za pomocą wywołania procedury *Plot*, a wyświetlanie odcinka za pomocą wywołania procedury *Draw*.

Procedura *Plot*

Wywołanie: *Plot*(*xCoord*,*yCoord*,*Color*)
Przyjmuje się, że xCoord i *yCoord* i *Color* są wyrażeniami typu *integer*.

Wykonanie procedury *Plot* powoduje wyświetlenie jednego piksela w tym punkcie ekranu, który ma współrzędne (*xCoord*,*yCoord*). Wyświetlanie odbywa się w kolorze określonym przez wyrażenie *Color*. W trybach *GraphColorMode i GraphMode* jest brana pod uwagę bieżąca paleta barw. W trybie *HiRes* dla argumentu *Color* o wartości 0 następuje wyświetlenie punktu w kolorze czarnym. Dla argumentu *Color* o wartości 1 następuje wyświetlenie punktu w kolorze określonym za pomocą procedury *HiResColor*.

Przykład

Plot(0,0,1)

• Wykonanie procedury powoduje wyświetlenie punktu w lewym-górnym narożniku ekranu.
• Jeśli wyświetlanie odbywa się w ramach palety nr 0, to w trybie *GraphColorMode* punkt ma kolor zielony, a w trybie *GraphMode* ma kolor niebieski. □

Procedura *Draw*

Wywołanie: *Draw*(*x1*,*y1*,*x2*,*y2*,*Color*) □
Przyjmuje się, że *x1*, *y1*, *x2*, *y2* i *Color* są wyrażeniami typu *integer*.

Wykonanie procedury *Draw* powoduje wyświetlenie odcinka linii prostej, zawartego między punktami ekranu o współrzędnych (*x1*, *y1*) i (*x2*, *y2*). Odcinek ten jest wyświetlany w kolorze określonym przez wyrażenie *Color*. W trybach *Graph ColorMode* i *GraphMode* jest brana pod uwagę bieżąca paleta barw. W trybie *HiRes* dla argumentu *Color* o wartości 0 wyświetlanie

odbywa się w kolorze czarnym, a dla argumentu *Color* o wartości 1 odbywa się w kolorze określonym za pomocą wywołania procedury *HiResColor*.

Przykład

Draw(0,0,639,199,1)

• Wykonanie przytoczonej procedury w trybie *HiRes* powoduje wyświetlenie głównej przekątnej ekranu. Jeśli odbywa się to po wywołaniu procedury *HiResColor* (*Red*), to przekątna ta jest czerwona. □

Procedura *GraphColorMode*

Wywołanie: GraphColorMode

Procedura *GraphColorMode* jest bezparametrowa.

Wykonanie procedury *GraphColorMode* powoduje ustanowienie trybu graficznego kolorowego 320 × 200 pikseli. W tym trybie współrzędne poziome pikseli przybierają wartości z przedziału 0..319, a współrzędne pionowe przybierają wartości z przedziału 0..199. Lewy-górny piksel ekranu ma współrzędne (0,0). Po wybraniu jednej z palet wyszczególnionych w tabl. 20.2 można uzyskać wyświetlanie punktów w dowolnym z 4 kolorów tej palety. Kolor tła można ustalić za pomocą procedury *GraphBackground*.

Przykład

```
program Dot;
begin
  GraphColorMode;
  Palette(1);
  GraphBackground(Blue);
  Plot(160,100,2);
  repeat until KeyPressed;
  TextMode
end.
```

• Wykonanie programu aż do instrukcji repeat wyłącznie, powoduje wyświetlenie w trybie graficznym kolorowym jednego punktu w środku ekranu. Cały ekran zostaje wypełniony kolorem tła — niebieskim, a punkt zostaje wyświetlony w kolorze 2 wybranej palety, tj. karmazynowym.

• Gdyby z programu usunięto wywołania procedur *Palette* i *GraphBackground*, to domniemaną paletą byłaby paleta nr 0, a domniemanym kolorem tla — czarny. W takim przypadku wspomniany punkt zostałby wyświetlony w kolorze 2 wybranej palety, tj. czerwonym.

• W programie posłużono się instrukcja repeat dla uniknięcia przedwczesnego wyczyszczenia ekranu. □

Procedura *GraphMode*

Wywolanie: GraphMode

Procedura GraphMode jest bezparametrowa.

Wykonanie procedury *GraphMode* powoduje włączenie trybu graficznego czarno-białego 320 × 200 pikseli. W tym trybie współrzędne poziome pikseli przybierają wartości z przedziału 0..319, a współrzędne pionowe przybierają wartości z przedziału 0..199. Lewy-górny piksel ekranu ma współrzędne (0,0). Na monitorze RGB, takim jak np. monitor kolorowy firmy IBM, możliwe jest wyświetlanie kolorowe. Po wybraniu jednej z 2 palet wyszczególnionych w tabl. 20.3 można uzyskać wyświetlanie punktów w dowolnym z 4 kolorów tej palety. Kolor tła można ustalić za pomocą procedury *GraphBackground.*

Przykład

```
program Dot;
begin
  GraphMode;
  Plot(160,100,2);
  repeat until KeyPressed;
  TextMode
end.
```

- Punkt zostanie wyświetlony na domniemanym tle, które jest czarne, w kolorze 2 domniemanej palety, tj. palety nr 0.
- Punkt zostanie wyświetlony w kolorze czerwonym.
- Gdyby wywołaniu procedury *Plot* nadano postać

Plot(160,100,*Red*)

to byłoby ono w zasadzie niepoprawne, gdyż nazwa *Red* reprezentuje literał o wartości 4, a w każdej palecie istnieją tylko kolory o numerach z przedziału 0..3.

- W istocie punkt zostałby wyświetlony w kolorze tła i byłby niewidoczny. □

Procedura *HiRes*

Wywołanie: *HiRes*

Procedura *HiRes* jest bezparametrowa.

Wykonanie procedury *HiRes* powoduje włączenie trybu graficznego czarno-białego 640 × 200 pikseli. W tym trybie, nazywanym także *trybem wysokiej rozdzielczości*, współrzedne poziome pikseli przybierają wartości z przedziału 0..199. Lewy-górny piksel ma współrzędne (0,0). Punkty są wyświetlane w jednym z kolorów wyszczególnionych w tabl. 20.1. Wybór koloru odbywa się za pomocą procedury *HiResColor*. Tło jest zawsze czarne.

Przykład

```
program Ewa;
begin
  HiRes;
  HiResColor(Red);
  Write('Ewa');
  Plot(18,5,1);
  repeat until KeyPressed;
  TextMode
end.
```

• Wykonanie przytoczonego programu aż do instrukcji repeat wyłącznie powoduje wyprowadzenie na ekran napisu *Ewa* oraz jednego punktu świecącego.
• Zarówno punkty tworzące napis, jak i wspomniany punkt świetlny są wyprowadzane w kolorze czerwonym.
• Współrzędne punktu są tak dobrane, że jest on wyświetlany we wnętrzu obszaru otoczonego „brzuszkiem" litery a.
• Gdyby z programu usunięto wywołanie procedury *HiResColor*, to wszystkie punkty zostałyby wyświetlone w kolorze domniemanym — białym. □

Procedura *HiResColor*

Wywołanie: *HiResColor(Hue)*

Przyjmuje się, że *Hue* jest wyrażeniem typu *integer* reprezentującym daną o wartości z przedziału 0..15. Związek wartości z kolorami wynika z tabl. 20.1.

Wykonanie procedury *HiResColor* powoduje określenie koloru punktów wyświetlanych w trybie graficznym wysokiej rozdzielczości. Wywołanie w tym trybie procedury *HiResColor* powoduje natychmiastową zmianę koloru wszystkich właśnie wyświetlanych punktów.

Przykład

```
program Secret;
begin
  HiRes;
  HiResColor(Black);
  Write('Jan');
  repeat until KeyPressed;
  Delay(10000);
  TextMode
end.
```

• Wykonanie procedury *Write* powoduje wyprowadzienie napisu *Jan*.
• Napis ten jest niewidoczny, ponieważ wyświetlanie odbywa się w kolorze tła.

• Po wprowadzeniu z klawiatury dowolnego znaku, np. spacji, na ekranie ujawnia się czerwony napis *Jan* i po 10 sekundach znika. □

Procedura *Palette*

Wywołanie: *Palette*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu integer.

Wykonanie procedury *Palette* powoduje wybranie palety o numerze określonym przez *Num*. W trybie *GraphColorMode* są dostępne palety o numerach z przedziału 0..1, przytoczone w tabl. 20.3.

Przykład

```
program ChangeColors;
begin
  GraphColorMode;
  Palette(1);
  Plot(0,0,1);
  Plot(199,319,2);
  repeat until KeyPressed;
  Palette(0)
  Delay(10000);
  TextMode
end.
```

• Wykonanie procedury *Plot* powoduje wyświetlenie dwóch punktów: jednego w kolorze turkusowym i drugiego w kolorze karmazynowym.
• Po wprowadzeniu z klawiatury dowolnego znaku następuje zmiana palety. Punkt w kolorze turkusowym zmienia kolor na zielony, a punkt w kolorze karmazynowym zmienia kolor na czerwony.
• Po upływie 10 sekund obraz znika. □

Procedura *GraphBackground*

Wywołanie: *GraphBackground*(*Hue*)

Przyjmuje się, że *Hue* jest wyrażeniem typu *integer*.
Wykonanie procedury *GraphBackground* powoduje określenie koloru tła na podstawie wartości danej reprezentowanej przez wyrażenie *Hue*. Odwzorowanie wartości liczbowych na kolory przytoczono w tabl. 20.1. Wykonanie omawianej procedury w trybie graficznym *HiRes* nie ma żadnych skutków. W trybie tekstowym następuje jedynie zmiana koloru obrzeża ekranu.

Przykład

```
program Vanish;
begin
  GraphColorMode;
  Plot(0,0,2);
  repeat until KeyPressed;
  GraphBackground(Red);
  Delay(10000);
  TextMode
end.
```

• Wykonanie procedury *Plot* powoduje wyświetlenie na czarnym tle jednego czerwonego punktu.

• Po wprowadzeniu z klawiatury dowolnego znaku następuje zmiana tła na czerwone. Powoduje to, że wyświetlony uprzednio punkt zlewa się z nowym tłem i przestaje być widoczny. □

Definiowanie okien

W normalnych warunkach wykonywanie operacji ekranowych może dotyczyć całego ekranu. W języku Turbo Pascal jest. ponadto możliwe definiowanie fragmentu ekranu jako okna i wykonywanie operacji ekranowych tak, jakby to okno zajmowało cały dostępny ekran.

Do definiowania okna tekstowego służy procedura *Window*, a do definiowania okna graficznego procedura *GraphWindow*. W każdej chwili może być zdefiniowane tylko jedno okno. Można przyjąć, że bezpośrednio po ustanowieniu trybu tekstowego albo graficznego zostaje niejawnie ustalone odpowiednio okno tekstowe albo graficzne. Kształt i rozmiary takiego okna pokrywają się z kształtem i rozmiarami ekranu. Należy nadmienić, że definiowane mogą być tylko okna prostokątne. Zdefiniowanie okna nie powoduje zmiany obrazu na ekranie.

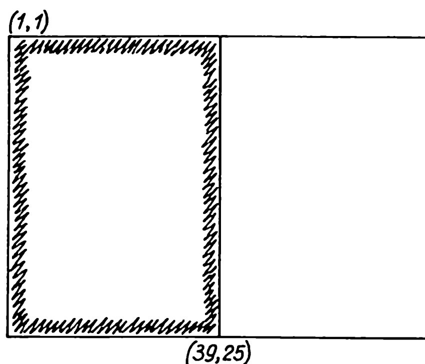
Na rysunku 20.1 przedstawiono przykładowe okno dla trybu tekstowego *BW80*, a na rys. 20.2 – przykładowe okno dla trybu graficznego wysokiej rozdzielczości. Położenie okien dobrano w taki sposób, aby stanowiły ono lewe połowy ekranu.

Procedura *Window*

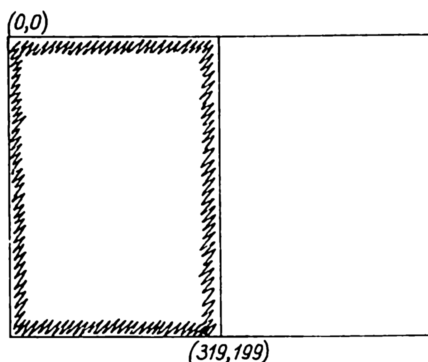
Wywołanie: *Window(xMin,yMin,xMax,yMax)*

Przyjmuje się, że *xMin, yMin, xMax* i *yMax* są wyrażeniami typu *integer*.

Wykonanie procedury *Window* powoduje zdefiniowanie okna tekstowego,



Rys. 20.1 Okno tekstowe



Rys. 20.2 Okno graficzne

którego lewy-górny narożnik ma współrzędne znakowe (*xMin,yMin*), a prawy-dolny narożnik ma współrzędne znakowe (*xMax,yMax*). Wymaga się, aby tak zdefiniowane okno miało co najmniej rozmiary 2×2 znaki. Można przyjąć, że w chwili ustanowienia trybów tekstowych *BW40* i *C40* zostaje niejawnie wykonana instrukcja

Window(1,1,40,25)

a w chwili ustanowienia trybów tekstowych *BW40* i *C80* zostaje niejawnie wykonana instrukcja

Window(1,1,80,25)

Po wykonaniu procedury *Window* wszystkie współrzędne ekranu występujące poza wywołaniem procedury *Window* są liczone względem nowego okna. Dotyczy to w szczególności wykonania procedury *GotoXY*.

Przykład

```
program JanAndEwa;
var
  Ch : char;
begin
  TextMode(BW40);
  Window(10,10,20,20);
  GotoXY(2,2);
  Write('Jan');
  Read(Kbd,Ch);
  Window(1,1,20,20);
  GotoXY(11,11);
  Write('Ewa');
  repeat until KeyPressed
end.
```

- Bezpośrednio po rozpoczęciu wykonywania programu zostaje ustanowione okno tekstowe domniemane (1,1,80,25).
- Podczas wykonywania instrukcji *TextMode*(BW40) zostaje ustanowione okno tekstowe (1,1,40,25).
- Po wykonaniu instrukcji *Window*(10,10,20,20) zostaje ustanowione okno tekstowe (10,10,20,20).
- Wykonanie przytoczonego programu powoduje wyprowadzenie napisu *Jan*, a po wprowadzeniu z klawiatury dowolnego znaku, zastąpienie trzech liter tego napisu trzema literami napisu *Ewa*. □

Procedura *GraphWindow*

Wywołanie: *GraphWindow*(*xMin,yMin,xMax,yMax*)

Przyjmuje się, że *xMin*, *yMin*, *xMax*, i *yMax* są wyrażeniami typu *integer*.

Wykonanie procedury *GraphWindow* powoduje zdefiniowanie okna graficznego, którego lewy-górny narożnik ma współrzędne pikselowe (*xMin,yMin*), a prawy-dolny narożnik ma współrzędne pikselowe (*xMax,yMax*). Można przyjąć, że w chwili ustanowienia trybu graficznego wysokiej rozdzielczości zostaje niejawnie wykonana instrukcja

GraphWindow(0,0,639,199)

a w chwili ustanowienia dowolnego z pozostałych trybów graficznych zostaje niejawnie wykonana instrukcja

GraphWindow(0,0,319,199)

Po wykonaniu procedury *GraphWindow* wszystkie współrzędne ekranu występujące poza wywołaniem procedury *Window* są liczone względem nowego okna, a wykreślaniu podlegają tylko te punkty, których współrzędne znajdują się wewnątrz okna.

Przykład

```
program Diagonal;
begin
  HiRes;
  HiResColor(White);
  GraphWindow(20,20,50,50);
  Draw(0,0,30,0,1);
  Draw(30,0,30,30,1);
  Draw(30,30,0,30,1);
  Draw(0,30,0,0,1);
  Draw(0,0,40,40,1);
  repeat until KeyPressed;
  TextMode
end.
```

• Po zdefiniowaniu okna o wymiarach 30×30 zostaje wykreślone jego obrzeże.

• Ostatnia instrukcja *Draw* służy do wykreślenia głównej przekątnej okna. Odcinek wyznaczony przez współrzędne (30,30) i (40,40) znajduje się poza oknem i nie jest wykreślany. Z tego powodu ostatnia instrukcja *Draw* jest wykonywana tak jak instrukcja

Draw(0,0,30,30,1) □

Rozszerzenia biblioteczne

W zakres oprogramowania grafiki podstawowej wchodzą jedynie procedury do przełączania monitora między trybami znakowymi i graficznymi, procedury do wykreślania punktów i odcinków oraz podprogramy do zarządzania kolorami oraz ustalania pozycji kursora w trybach tekstowych.

Pozostałe podprogramy graficzne nie są już integralnie związane z językiem Turbo Pascal i pochodzą z bibliotek, które muszą być jawnie wskazane podczas kompilowania programu.

Wraz z systemem Turbo Pascal dla IBM PC jest dostarczana biblioteka źródłowa GRAPH.P zawierająca deklaracje podprogramów graficznych zawartych w bibliotece GRAPH.BIN. Podprogramy tej biblioteki są napisane w asemblerze, czemu zawdzięczają swoją dużą efektywność.

• W celu posłużenia się biblioteką graficzną zawartą w zbiorze GRAPH.BIN należy dokonać włączenia jej deklaracji. Odbywa się to za pomocą dyrektywy

{$*i GRAPH.P* }

Dyrektywa ta może wystąpić w dowolnym miejscu, w którym mogą wystąpić deklaracje podprogramów, ale musi poprzedzać odwołania do podprogramów z biblioteki GRAPH.BIN.

Argumentami szeregu podprogramów biblioteki GRAPH.BIN są wyrażenia określające kolor obiektu, np. punktu, łuku albo okręgu. Przyjmuje się, że jeżeli takie wyrażenie ma wartość nieujemną, to określa numer barwy bieżącej palety. Jeśli ma wartość −1, to kolor punktów tworzących obiekt wyniknie z rozpatrzenia kolorów tych punktów ekranu na których obiekt zostanie wykreślony.
Bezpośrednio po ustanowieniu trybu graficznego obowiązuje ustalenie, że wykreślanie w „kolorze" −1 jest wykreślaniem w kolorze punktów ekranu. Ustalenie to może być zmienione za pomocą procedury *ColorTable*. Procedura ta zostanie omówiona jako pierwsza, ponieważ z oferowanych przez nią możliwości można korzystać w większości z pozostałych procedur.

Procedura *ColorTable*

Wywołanie: *ColorTable(Hue 0,Hue 1,Hue 2,Hue 3)*

Przyjmuje się, że *Hue0, Hue1, Hue2* i *Hue3* są wyrażeniami typu *integer.*

Wykonanie procedury *ColorTable* powoduje zdefiniowanie wektora barw, branego pod uwagę podczas wyświetlania punktów w „kolorze" −1. W ogólnym przypadku argument Hue_i określa, na jaki kolor ma być zmieniony kolor punktu wyświetlanego w kolorze *i.* W szczególności dla trybu *HiRes*, wykonanie procedury

ColorTable(1,0,1,0)

powoduje, że wyświetlenie punktu w pewnym miejscu ekranu zmienia kolor tego punktu z koloru tła na kolor określony przez procedurę *HiResColor* i odwrotnie.

Przykład

```
program ColorSwitch;
{$i Graph.p }
var
  i : byte;
begin
  GraphColorMode;
  Palette(1);
  for i:= 0 to 10 do
    Draw(i,0,319,199−i,2);
  ColorTable(3,2,1,0);
  Draw(319,0,0,199,−1);
  repeat until KeyPressed;
  TextMode
end.
```

● Ponieważ w programie *ColorSwitch* odwołano się do procedury *ColorTable* należącej do zbioru *GRAPH.BIN*, konieczne było użycie dyrektywy włączenia zbioru.
● Wykonanie pierwszej instrukcji cyklu powoduje wykreślenie paska wzdłuż głównej przekątnej ekranu. Pasek ten jest wykreślany w kolorze karmazynowym, gdyż obowiązuje jeszcze domniemanie

ColorTable(0,1,2,3)

● Wykonanie drugiej procedury *Draw* powoduje wykreślenie drugiej przekątnej ekranu. Ponieważ odbywa się to po wykonaniu instrukcji

ColorTable(3,2,1,0)

przekątna ta jest wykreślana w kolorze jasnoszarym, z wyjątkiem punktu przecięcia z przekątną główną, który jest wykreślany w kolorze turkusowym. □

Procedura *Arc*

Wywołanie: *Arc*(*xCoord*,*yCoord*,*Angle*,*Radius*,*Color*)

Przyjmuje się, że *xCoord*, *yCoord*, *Angle*, *Radius* i *Color* są wyrażeniami typu *integer*.

Wykonanie procedury *Arc* powoduje wykreślenie łuku okręgu o promieniu *Radius* i w kolorze *Color*. Wykreślanie łuku rozpoczyna się w punkcie o współrzędnych (*xCoord*,*yCoord*). Argument *Angle* określa kąt łuku wyrażony w stopniach. Jeśli *Angle* > 0, to łuk jest wykreślany w kierunku zgodnym z ruchem wskazówek zegara. Jeśli *Angle* < 0, to jest wykreślany w kierunku przeciwnym. Jeśli argument *Color* reprezentuje daną o wartości −1, to kolor poszczególnych punktów łuku jest określony poprzez wektor barw.

Przykład

```
program Quadrant;
{$i Graph.p }
begin
  HiRes;
  HiResColor(Red);
  Arc(0,199,-180,100,1)
  repeat until KeyPressed;
  TextMode
end.
```

• Wykonanie procedury *Arc* powoduje wykreślenie półokręgu opartego na lewej krawędzi ekranu jako na średnicy. Półokrąg jest wykreślany w kolorze czerwonym. □

Procedura *Circle*

Wywołanie: *Circle*(*xCoord*,*yCoord*,*Radius*,*Color*)

Przyjmuje się, że *xCoord*, *yCoord*, *Radius* i *Color* są wyrażeniami typu *integer*.

Wykonanie procedury *Circle* powoduje wykreślenie okręgu o promieniu *Radius* i środku w punkcie o współrzędnych (*xCoord*,*yCoord*). Okrąg jest wykreślany w kolorze *Color* i w trybach graficznych 320 × 200 pikseli ma taki sam rozmiar w kierunku pionowym i poziomym. W trybie 640 × 200 pikseli ma

natomiast postać elipsy. Jeśli argument *Color* reprezentuje daną o wartości −1, to kolor poszczególnych punktów ekranu jest określony poprzez wektor barw.

Przykład

```
program PerfectCircle;
{$i Graph.p }
begin
  GraphMode;
  Circle(99,99,99,1);
  repeat until KeyPressed;
  TextMode
end.
```

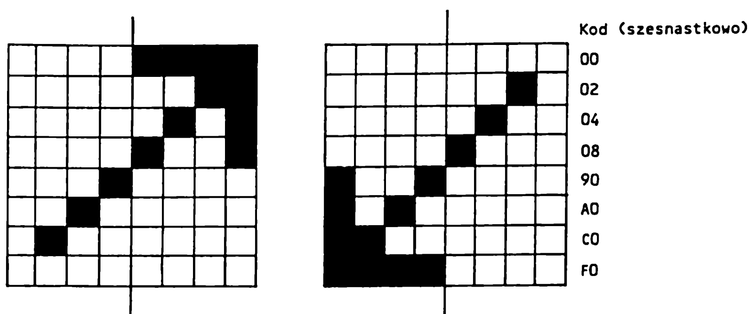
• Wykonanie procedury *Circle* powoduje wykreślenie największego okręgu, jaki bez obcięć mieści się na ekranie. Okrąg ten jest styczny do lewej oraz górnej krawędzi ekranu. □

Procedura *Pattern*

Wywołanie: *Pattern*(*Vector*)

Przyjmuje się, że *Vector* jest nazwą tablicy typu **array**[0..7] **of** *byte*.

Wykonanie procedury *Pattern* powoduje ustalenie wzoru wykorzystywanego do wypełniania obszaru przez procedurę *FillPattern*. Wzór ma postać tablicy 8 × 8 bitów i pojawia się na ekranie lustrzanie odbity zarówno w osi pionowej jak i poziomej. Z tego powodu, np. dla uzyskania na ekranie grupy strzałek zwróconych w kierunku prawego-górnego narożnika ekranu należy wyznaczyć kody wspomnianej tablicy 8 × 8 ze strzałką zwróconą w kierunku lewego-dolnego rogu ekranu, tak jak pokazano to na rys. 20.3.



Rys. 20.3 Projektowanie wzoru

Przykład

```
program Arrows;
{$i Graph.p }
const
  Arrow : array[0..7] of byte =
            ($00,$02,$04,$08,$90,$a0,$c0,$f0);
begin
  HiRes;
  Pattern(Arrow);
  FillPattern(0,0,639,199,1);
  repeat until KeyPressed;
  TextMode
end.
```

• Wykonanie instrukcji programu *Arrows* aż do instrukcji repeat wyłącznie powoduje wyświetlenie na ekranie strzałek skierowanych w stronę prawego-górnego narożnika ekranu. □

Procedura *FillPattern*

Wywołanie: *FillPattern(xMin,yMin,xMax,yMax,Color)*

Przyjmuje się, że *xMin*, *yMin*, *xMax*, *yMax* i *Color* są wyrażeniami typu *integer*.

Wykonanie procedury *FillPattern* powoduje wypełnienie prostokątnego obszaru wyznaczonego przez współrzędne (*xMin*,*yMin*) i (*xMax*,*yMax*) wzorem w kolorze o numerze *Color*, określonym przez procedurę *Pattern*. Wzór do wypełniania obszaru ma rozmiary 8 × 8 pikseli. Po umieszczeniu w lewym-dolnym rogu wypełnianego obszaru jest on powielany od dołu do góry i od lewej do prawej. Te bity wzoru, które mają wartość 0, nie powodują zmiany koloru punktów ekranu.

Przykład

```
program Stripes;
{$i Graph.p }
const
  PatternArray : array[0..7] of byte =
                 ($99,$99,$99,$99,$99,$99,$99,$99);
begin
  GraphColorMode;
  FillScreen(2);
  Pattern(PatternArray);
```

```
  FillPattern(2,0,5,199,3);
  repeat until KeyPressed;
  TextMode
end.
```

• Wykonanie procedury *FillScreen* powoduje wypełnienie całego ekranu punktami w kolorze czerwonym.
• Wykonanie procedury *Pattern* powoduje zdefiniowanie wzoru mającego postać trzech pasków pionowych, z których środkowy jest szerszy.
• Wykonanie procedury *FillPattern* powoduje pojawienie się na ekranie dwóch pasków pionowych w kolorze brązowym. Paski są tej samej szerokości, ponieważ szerszy podlega obcięciu na pionowej granicy obszaru. □

Procedura *FillScreen*

Wywołanie: *FillScreen*(*Color*)

Przyjmuje się, że Color jest wyrażeniem typu *integer*.

Wykonanie procedury *FillScreen* powoduje wypełnienie aktywnego okna graficznego punktami w kolorze *Color*. Jeśli argument *Color* reprezentuje daną o wartości −1, to kolor poszczególnych punktów wypełniających okno jest określony przez wektor barw.

Przykład

```
program InvertScreen;
{$i Graph.p }
begin
  GraphColorMode;
  Draw(0,0,319,199,2);
  Delay(5000);
  ColorTable(3,2,1,0);
  FillScreen(—1);
  repeat until KeyPressed;
  TextMode
end.
```

• Wykonanie procedury *Draw* powoduje wykreślenie głównej przekątnej ekranu w kolorze czerwonym.
• Po upływie 5 sekund następuje inwersja kolorów określona przez wektor barw. Przekątna przybiera kolor zielony, a pozostała część ekranu kolor brązowy. □

Procedura *FillShape*

Wywołanie: *FillShape(xCoord,yCoord,FillColor,BorderColor)*

Przyjmuje się, że *xCoord*, *yCoord*, *FillColor* i *BorderColor* są wyrażeniami typu *integer*.

Wykonanie procedury *FillShape* powoduje zlokalizowanie punktu o współrzędnych (*xCoord,yCoord*), a następnie wypełnienie obszaru otaczającego ten punkt — punktami w kolorze *FillColor*. Zakłada się, że obrzeże obszaru otaczającego wspomniany punkt ma kolor *BorderColor*. Wymaga się, aby kolor wypełniający nie był kolorem tła.

Przykład

```
program FillUp;
{$i Graph.o }
begin
  HiRes;
  GotoXY(2,1);
  Write('Ewa');
  FillShape(2,2,1,1);
  repeat until KeyPressed;
  TextMode
end.
```

• Wykonanie procedury *FillShape* powoduje takie zapełnienie ekranu, że pozostaną na nim tylko trzy czarne punkty. Stanowią one wnętrze litery a pochodzącej ze słowa *Ewa*.

• Wypełnienie całego ekranu wynika stąd, iż można przyjąć, że zewnętrzne obrzeże obszaru występuje poza ekranem, a więc ekran wypełniany jest aż do jego brzegów. □

Procedura *GetPic*

Wywołanie: *GetPic(Buffer,xMin,yMin,xMax,yMax)*

Przyjmuje się, że *Buffer* jest nazwą zmiennej dowolnego typu, natomiast *xMin*, *yMin*, *xMax* i *yMax* są wyrażeniami typu *integer*.

Wykonanie procedury *GetPic* powoduje skopiowanie zawartości pamięci ekranu związanej z prostokątem o współrzędnych przeciwległych narożników (*xMin*, *yMin*) i (*xMax*, *yMax*) do obszaru pamięci operacyjnej zajmowanego przez zmienną *Buffer*. Wymaga się, aby rozmiar obszaru przydzielonego zmiennej *Buffer* był nie mniejszy niż to wynika z przytoczonych poniżej wzorów:

dla trybów 320 × 200 pikseli:

((*Hor*+3) **div** 4) * *Ver* * 2+6

dla trybu 640 × 200 pikseli:

((*Hor*+7) **div** 8) * *Ver*+6

gdzie:

Hor = *abs*(*xMin* − *xMax*) +1
Ver = *abs*(*yMin* − *yMax*) +1

Sposób reprezentowania obrazu w zmiennej *Buffer* jest taki, że pierwsze 3 pary bajtów stanowią nagłówek obrazu, a pozostałe zawierają dane o obrazie. Nagłówek zawiera kolejno: określenie trybu graficznego (1 dla 640 × 200; 2 dla 320 × 200), szerokość obrazu i wysokość obrazu. Szerokość jest zaokrąglona do pełnych bajtów. Kopiowanie zawartości pamięci ekranu odbywa się wierszami, od wiersza najniższego do najwyższego i od lewej do prawej. Skrajne-lewe piksele wierszy stają się najbardziej znaczącymi bitami bajtów danych.

Przykład

```
program Save;
{$i Graph.p }
var
  SaveArea : record
               Mode : integer;
               Hor, Ver : integer;
               Data : array[0..199] of byte
             end.
begin
  HiRes;
  Draw(0,0,0,199,1);
  GetPic(SaveArea,0,0,0,199);
  ...
end.
```

Wykonanie procedury *GetPic* jest w rozpatrywanym kontekście równoważne wykonaniu instrukcji

```
with SaveArea do begin
  Mode := 1;
  Hor := 1;
  Ver := 200;
  for i := 0 to 199 do
    Data[i] := $80
end
```

□

Procedura *PutPic*

Wywołanie: *PutPic(Buffer,xCoord,yCoord)*

Przyjmuje się, że *Buffer* jest nazwą zmiennej dowolnego typu, natomiast *xCoord* i *yCoord* są wyrażeniami typu *integer*.

Wykonanie procedury *PutPic* powoduje umieszczenie na ekranie prostokątnego obrazu zapamiętanego za pomocą procedury *GetPic*. Obraz jest umieszczony na ekranie w taki sposób, że jego lewy-dolny narożnik zajmuje pozycję o współrzędnych (*xCoord,yCoord*). Ustalanie koloru wyświetlanych bitów odbywa się przez wektor barw.

Przykład

```
program CopyLine;
{$i Graph.p }
var
  SaveArea : array[1..206] of byte:
begin
  HiRes;
  Draw(0,0,0,199,1);
  GetPic(SaveArea,0,0,0,199);
  PutPic(SaveArea,639,199);
  repeat until KeyPressed;
  TextMode
end.
```

• Wykonanie procedury *PutPic* ma w rozpatrywanym kontekście taki sam skutek jak wykonanie procedury

Draw(639,0,639,199,1) □

Funkcja *GetDotColor*

Wywołanie: *GetDotColor(xCoord,yCoord)*

Przyjmuje się, że *xCoord* i *yCoord* są wyrażeniami typu *integer*.

Rezultatem funkcji *GetDotColor* jest dana typu *integer* określająca kolor punktu ekranu o współrzędnych (*xCoord,yCoord*). Kolor jest określany w ramach bieżącej palety. Jeśli współrzędne punktu wykraczają poza okienko, to rezultatem funkcji jest dana o wartości −1.

Przykład

```
program Color;
{$i Graph.p }
```

```
begin
  GraphColorMode;
  Palette(1);
  ColorTable(2,1,0,3);
  Plot(0,0,-1);
  Delay(5000);
  Palette(0);
  GotoXY(1,1);
  Write(GetDotColor(0,0))
  repeat until KeyPressed;
  TextMode
end.
```

- Punkt o współrzędnych (0,0) zostaje wyświetlony w kolorze karmazynowym. Po upływie 5 sekund jego kolor zmienia się na czerwony.
- Wykonanie procedury *Write* powoduje wyprowadzenie liczby 2. □

Grafika żółwiowa

Idea *grafiki żółwiowej* wiąże się z pojęciem „żółwia", który poruszając się po ekranie kreśli odcinki linii prostych. Ponieważ odcinki te mogą być dowolnie krótkie i niemal dowolnie zorientowane, nic nie stoi na przeszkodzie, aby figury kreślone przez żółwia miały nie tylko postać łamanych, ale również postać dowolnych aproksymowanych przez nie linii krzywych.

Wykonywanie wykresów żółwiowych jest możliwe w dowolnym z trybów graficznych i wymaga — jak uprzednio — zbioru *GRAPH.P*. Same procedury żółwiowe znajdują się oczywiście w zbiorze *GRAPH.BIN*.

Na ekranie monitora żółw jest przedstawiony w postaci małego trójkąta, który może być dowolnie przesuwany i obracany. Bezpośrednio po aktywowaniu trybu graficznego trójkąt ten jest niewidoczny, jednak istnieje procedura, której wykonanie powoduje przywołanie żółwia na ekran. Ma on wówczas kolor nr 3 bieżącej palety barw.

Wykresy żółwiowe dokonują się w obszarze ekranu, który będzie tu nazywany *oknem żółwiowym*. Bezpośrednio po ustanowieniu trybu graficznego takim oknem jest cały ekran. Zakres dostępnych współrzędnych zależy wówczas od aktualnego trybu graficznego zgodnie z następującym zestawieniem:

tryb 320 × 200 pikseli:

$x = -159..160;\ y = -99..100$

tryb 640 × 200 pikseli:

$x = -319..320;\ y = -99..100$

Poczatek układu współrzędnych żółwiowych znajduje się zawsze w środku okna. Bezpośrednio po ustanowieniu trybu graficznego żółw znajduje się w punkcie (0,0) i jest skierowany do góry.

Z żółwiem jest związane *pióro*, które może być podniesione albo opuszczone. Jeśli jest opuszczone, to podczas ruchu żółwia w obszarze okna, kreśli ono linię prostą.

Bezpośrednio po ustanowieniu okna żółwiowego, które przez domniemanie pokrywa się z oknem graficznym, przemieszczenie żółwia poza okno czyni go niewidocznym. Widoczność żółwia można przywrócić sprowadzając go do obszaru okna albo wykonując procedurę *Wrap*. Po jej wykonaniu wszystkie współrzędne żółwiowe będą brane modulo rozmiary okna, co m.in. spowoduje, że przekroczenie przez żółwia np. górnej krawędzi okna uczyni go widocznym w pobliżu krawędzi dolnej.

Liczba podprogramów umożliwiających realizowanie grafiki żółwiowej jest dość znaczna. Te, które zostaly zrealizowane w implementacji dla IBM PC zostaną tu przytoczone w porządku alfabetycznym.

Procedura *Back*

Wywołanie: *Back*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Wykonanie procedury *Back* powoduje przemieszczenie żółwia w kierunku „do tyłu" o *Num* pikseli. Jeśli *Num* reprzentuje daną o wartości ujemnej, to żółw przemieszcza się do przodu. Jeśli pióro związane z żółwiem jest opuszczone, to przemieszczanie się żółwia może powodować kreślenie linii.

Procedura *ClearScreen*

Wywołanie: *ClearScreen*

Procedura *ClearScreen* jest bezparametrowa.

Wykonanie procedury *ClearScreen* powoduje wyczyszczenie aktywnego okna i przemieszczenie żółwia do pozycji o współrzędnych (0,0), znajdującej się w środku okna. Po wykonaniu tej operacji żółw jest niewidoczny, a pióro związane z żółwiem — opuszczone.

Procedura *Forwd*

Wywołanie: *Forwd*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Wykonanie procedury *Forwd* powoduje przemieszczenie żółwia w kierunku „do przodu" o *Num* pikseli. Jeśli *Num* reprezentuje daną o wartości ujemnej, to żółw przemieszcza się do tyłu. Jeśli pióro związane z żółwiem jest opuszczone, to przemieszczanie się żółwia może powodować kreślenie linii.

Funkcja *Heading*

Wywołanie: *Heading*

Funkcja *Heading* jest bezparametrowa.

Rezultatem funkcji *Heading* jest dana typu *integer*, której wartość określa kierunek, w którym jest zwrócony żółw. Kierunek jest określany z dokładnością do 1 stopnia. Wartość 0 oznacza kierunek „do góry", a kolejne wartości dodatnie, aż do 359 włącznie, oznaczają kierunki tworzone przez obrót zgodnie z ruchem wskazówek zegara.

Procedura *HideTurtle*

Wywołanie: *HideTurtle*

Procedura *HideTurtle* jest bezparametrowa.

Wykonanie procedury *HideTurtle* powoduje, że żółw staje się niewidoczny. Niewidoczność żółwia nie pozbawia go jednak innych właściwości, jak np. zdolności do przemieszczania się albo do kreślenia linii.

Procedura *Home*

Wywołanie: *Home*

Procedura *Home* jest bezparametrowa.

Wykonanie procedury *Home* powoduje umieszczenie żółwia w pozycji początkowej, tj. w punkcie o współrzędnych (0,0) i nadanie mu kierunku „do góry". Zmiana pozycji żółwia nigdy nie powoduje kreślenia linii.

Procedura *NoWrap*

Wywołanie: *NoWrap*

Procedura *NoWrap* jest bezparametrowa.

Wykonanie procedury *NoWrap* powoduje, że przemieszczenie żółwia poza okno czyni go niewidocznym.

Procedura *PenDown*

Wywołanie: *PenDown*

Procedura *PenDown* jest bezparametrowa.

Wykonanie procedury *PenDown* powoduje opuszczenie pióra związanego z żółwiem. Spowoduje to, że przemieszczanie się żółwia będzie w obrębie ekranu powodować kreślenie linii.

Procedura *PenUp*

Wywołanie: *PenUp*

Procedura *PenUp* jest bezparametrowa.

Wykonanie procedury *PenUp* powoduje podniesienie pióra związanego z żółwiem. Spowoduje to, że przemieszczanie się żółwia nie będzie powodować kreślenia linii.

Procedura *SetHeading*

Wywołanie: *SetHeading*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Wykonanie procedury *SetHeading* powoduje zwrócenie żółwia w kierunku określonym przez *Num*. Kierunek jest określany z dokładnością do 1 stopnia. Wartość 0 oznacza kierunek „do góry", a kolejne wartości dodatnie, aż do 359 włącznie, oznaczają kierunki tworzone przez obrót zgodnie z ruchem wskazówek zegara.

Procedura *SetPenColor*

Wywołanie: *SetPenColor*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Wykonanie procedury *SetPenColor* powoduje wybranie koloru używanego do kreślenia linii. Kolor jest wybierany z bieżącej palety barw, zgodnie z zasadami jak dla odpowiednich trybów graficznych. W szczególności posłużenie się kolorem reprezentowanym przez −1 powoduje odwołanie się do wektora barw.

Procedura *SetPosition*

Wywołanie: *SetPosition*(*xCoord*, *yCoord*)

Przyjmuje się, że *xCoord* i *yCoord* są wyrażeniami typu *integer*.

Wykonanie procedury *SetPosition* powoduje umieszczenie żółwia w pozycji o współrzędnych (*xCoord,yCoord*). Zmiana pozycji żółwia nigdy nie powoduje kreślenia linii. Wykonanie procedury *SetPosition*(0,0) jest równoważne wykonaniu procedury *Home*.

Procedura *ShowTurtle*

Wywołanie: *ShowTurtle*

Procedura *ShowTurtle* jest bezparametrowa.

Wykonanie procedury *ShowTurtle* powoduje, że żółw staje się widoczny.

Procedura *TurnLeft*

Wywołanie: *TurnLeft*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Wykonanie procedury *TurnLeft* powoduje obrót żółwia „w lewo" o liczbę stopni określoną przez *Num*. Jeśli *Num* reprezentuje daną o wartości ujemnej, to obrót następuje w prawo.

Procedura *TurnRight*

Wywołanie: *TurnRight*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Wykonanie procedury *TurnRight* powoduje obrót żółwia „w prawo" o liczbę stopni określoną przez *Num*. Jeśli *Num* reprezentuje daną o wartości ujemnej, to obrót następuje w lewo.

Procedura *TurtleWindow*

Wywołanie: *TurtleWindow*(*xCooord,yCoord,Hor,Ver*)

Przyjmuje się, że *xCoord*, *yCoord*, *Hor* i *Ver* są wyrażeniami typu *integer*.

Wykonanie procedury *TurtleWindow* powoduje utworzenie okna żółwiowego. Środek tego okna znajduje się w punkcie o współrzędnych (*xCoord,yCoord*), a okno liczy *Hor* pikseli w poziomie i *Ver* pikseli w pionie. Bezpośrednio po ustanowieniu trybu graficznego oknem żółwiowym jest cały ekran. Oznacza to, że w trybach 320 × 200 pikseli zostaje niejawnie wykonana procedura

TurtleWindow(159,99,320,200)

a w trybie 640 × 200 pikseli zostaje niejawnie wykonana procedura

TurtleWindow(319,99,640,200)

Funkcja *TurtleThere*

Wywołanie: *TurtleThere*

Funkcja *TurtleThere* jest bezparametrowa.

Rezultatem funkcji *TurtleThere* jest dana typu *boolean* wyrażająca prawdziwość zdania „żółw jest widoczny".

Procedura *Wrap*

Wywołanie: *Wrap*

Procedura *Wrap* jest bezparametrowa.

Wykonanie procedury *Wrap* powoduje, że przemieszczenie się żółwia poza okno jest traktowane tak, jakby żółw znalazł się w punkcie o współrzędnych wziętych modulo rozmiary okna.

Funkcja *xCor*

Wywołanie: *xCor*

Funkcja *xCor* jest bezparametrowa.

Rezultatem funkcji *xCor* jest dana typu *integer*, określająca odciętą bieżącej pozycji żółwia.

Funkcja *yCor*

Wywołanie: *yCor*

Funkcja *yCor* jest bezparametrowa.

Rezultatem funkcji *yCor* jest dana typu *integer*, określająca rzędna bieżącej pozycji żółwia.

Przykład

```
program Polygon;
{$i Graph.p }
var
  aStep,i,Num : byte;
begin
  Read(Num);
  while Num > 2 do begin
    GraphColorMode;
```

```
        PenUp;
        Back(50)
        PenDown;
        TurnLeft(90);
        aStep := trunc(360 / Num);
        for i := 1 to Num do begin
          Forwd(10);
          TurnLeft(aStep)
        end;
        repeat until KeyPressed;
        TextMode;
        Read(Num)
    end
end.
```

- Przytoczony program służy do wykreślania wielokątów foremnych.
- Instrukcje *Read* służą do wprowadzenia liczby boków wielokątów.
- Zakończenie wykonywania programu następuje po wprowadzeniu liczby boków mniejszej od 3. □

Dźwięk

Środki umożliwiające uzyskanie dźwięku są w IBM PC bardzo skromne. Sprowadzają się one do wywołania dźwięku o ustalonej częstotliwości albo wyciszenia go.

Procedura *Sound*

Wywołanie: *Sound*(*Num*)

Przyjmuje się, że *Num* jest wyrażeniem typu *integer*.

Wykonanie procedury *Sound* powoduje wywołanie jednostajnego dźwięku o częstotliwości *Num* herców.

Procedura *NoSound*

Wywołanie: *NoSound*

Procedura *NonSound* jest bezparametrowa.
Wykonanie procedury *NoSound* powoduje całkowite wyciszenie dźwięku wywołanego za pomocą procedury *Sound*.

Przykład

```
program Note400;
begin
  Sound(400);
  Delay(2000);
  NoSound
end.
```

• Wykonanie przytoczonego programu powoduje wydanie dźwięku o częstotliwości 400 herców, trwającego 2 sekundy. □

21. Przykłady programów

Przytoczone tu programy przykładowe zostały uruchomione na mikrokomputerze IBM PC. Ilustrują one posługiwanie się wybranymi podprogramami języka Turbo Pascal, demonstrując jego przydatność do przetwarzania tekstów i programowania grafiki.

Program *CopyDocuments*

Wykonanie programu *CopyDocuments*, przedstawionego na wydruku 21.1, w prawo od kolumny zawierającej komentarze, powoduje utworzenie czytelnej kopii dowolnego tekstu źródłowego, znajdującego się we wskazanym zbiorze. Kopię taką uzyskuje się przez wielokrotne nadrukowywanie poszczególnych wierszy tekstu. Ponadto, każdy wyprowadzany wiersz jest opatrywany numerem porządkowym zawartym w nawiasach klamrowych. W szczególności wydruk 21.1 został uzyskany za pomocą programu *CopyDocuments*.

Wiersze programu od 26 do 44 służą do określenia nazwy zbioru zawierającego tekst źródłowy, określenia liczby nadrukowań wiersza i rozstrzgnięcia, czy wysuwanie papieru do nowej strony ma być ręczne czy automatyczne. W tym drugim przypadku, wysunięcie papieru będzie realizowane za pomocą wysłania do drukarki znaku sterującego ^L. Odwolanie do takiego znaku występuje w procedurze *CheckPage*, w wieszu 23.

W ramach instrukcji repeat, rozpoczynającej się w wierszu 46, wyprowadzane są kolejne kopie tekstu źródłowego. Po wyprowadzeniu każdej kolejnej kopii na ekranie pojawia się informacja o liczbie już wyprowadzonych kopii i zapytanie, czy wyprowadzanie ma być kontynuowane. Jeśli podczas wyprowadzania kopii zostanie z klawiatury wprowadzony dowolny znak, to spowoduje to przerwanie wyprowadzania tej kopii.

Te wiersze tekstu, które rozpoczynają się od znaku (kropka) są ignorowane i nie pojawiają się na wydruku. Zrobiono to z myślą o tekstach przygotowanych za pomocą edytora WordStar, zawierających jego dyrektywy kropkowe. Pozostałe wiersze, o ile nie są puste, są opatrywane kolejnymi numerami umieszczonymi w nawiasach klamrowych. Opatrywanie wyprowadzanych wierszy takimi komentarzami jest realizowane w wierszach od 69 do 76 Nadrukowywanie wierszy jest natomiast realizowane w wierszach 78 i 79. Polega ono tym, że każdy wielokrotnie nadrukowany wiersz jest kończony jedynie znakiem powrotu karetki ^M.

```
{1}   program JanB;

{2}   begin

{3}     Writeln('Hello, I am JanB')

{4}   end.
```

Rys. 21.1 Wykorzystanie programu kopiującego

Wydruk 21.1 Program *CopyDocuments*

```
 {1}   program CopyDocuments;

 {2}   var
 {3}     LineBuf : string[132];
 {4}     OneChar : char;
 {5}     NewPage : boolean;
 {6}     Count,i : integer;
 {7}     Name : string[30];
 {8}     Inp,Out : text;
 {9}     Len : integer;
{10}     Check : boolean;
{11}     Tally : byte;
{12}     ManualPageEject : boolean;
{13}     LineNo : integer;
{14}     LineNoStr : string[12];

{15}   procedure CheckPage;
{16}   begin
{17}     if ManualPageEject then begin
{18}       GotoXY(1,25);
{19}       Write('Check page alignment, press any key');
{20}       Read(Kbd,OneChar);
{21}       GotoXY(1,25);
{22}       Write('':35)
{23}     end else Write(Out,^L);
{24}   end;

{25}   begin

{26}     ClrScr;
{27}     Writeln('Filename :-  ');
{28}     GotoXY(13,1);
{29}     Readln(name);
{30}     GotoXY(1,2);
{31}     Writeln('Number of strikes :-  ');
```

```
    Read(Kbd,OneChar);
    Count := ord(OneChar) - ord('0');
    if (Count < 1) or (Count > 9) then Count := 2;
    gotoXY(22,2);
    Write(chr(Count + ord('0')));
    gotoXY(1,3);
    Writeln('Manual Page Eject :-  ');
    repeat
      Read(Kbd,OneChar);
    until (OneChar = 'y') or (OneChar = 'n');
    gotoXY(22,3);
    Write(OneChar);
    ManualPageEject := OneChar = 'y';
    Tally := 0;
    repeat
      LineNo := 0;
      Assign(Inp,Name);
      Assign(Out,'LST:');
      Reset(Inp);
      Rewrite(Out);
      while not Eof(Inp) and not KeyPressed do begin
        NewPage := false;
        LineBuf := '';
        while not Eoln(Inp) and not NewPage do begin
          Read(Inp,OneChar);
          if OneChar <> ^L then
            LineBuf := LineBuf + OneChar
          else
            NewPage := true
        end;
        if Eoln(Inp) then Readln(Inp);
        Len := Length(LineBuf);
        if Len > 1 then
          while (Len > 1) and
                (Copy(LineBuf,Len,1) = ' ') do
            Len := Len - 1;
        if Len > 0 then
          if (Copy(LineBuf,1,1) <> '.') and
             (LineBuf <> ' ') then begin
            LineNo := (LineNo + 1) mod 1000;
            Str(LineNo:0,LineNoStr);
            LineNoStr := Copy('    ',1,
                              4 - Length(LineNoStr)) +
                              '{' + LineNoStr + '}';
            LineBuf := LineNoStr + '   ' + LineBuf;
            Len := Length(LineBuf);
            for i := 1 to Count do
              Write(Out,Copy(LineBuf,1,Len),^M)
          end;
        Writeln(Out);
        if NewPage then CheckPage
      end;
      Close(Inp);
      CheckPage;
      Close(Out);
      GotoXY(1,25);
      Tally := Tally + 1;
      if Tally = 1 then
        Write(' 1 copy printed.')
      else
        Write(Tally : 2,' copies printed.');
      GotoXY(20,25);
      Write('Copy next? ');
      Read(OneChar);
      while (OneChar <> 'y') and (OneChar <> 'n') do
        Read(OneChar);
```

```
      if OneChar = 'y' then begin
        GotoXY(1,25);
        Write('':31);
      end;
    until OneChar = 'n';
    ClrScr;
    Writeln('Done')

  end.
```

Program *BullsEye*

Program ten, przedstawiony na wydruku 21.2, demonstruje wykorzystanie prostych podprogramów graficznych. Rezultatem jego wykonania jest rysunek 21.2 zawierający szereg koncentrycznych okręgów. Niektóre z obszarów między okręgami są wypełnione. Okręgi są wykreślane za pomocą procedury *DrawRings.* Dwie grupy okręgów są wykreślone w oknie domniemanym, a jedna w konie ustanowionym za pomocą procedury *GraphWindow.* W obu tych przypadkach występuje obcinanie wykresu na obrzeżach okna. Ponieważ kolor użyty do wypełniania obszarów jest taki sam jak kolor okręgów i obrzeży okien, wypełnianie niektórych obszarów koncentrycznych kończy się na obrzeżu okna albo na łuku okręgu.

Wydruk 21.2 Program *BullsEye*

```
program BullsEye;

{$i Graph.p }

procedure DrawBorder(xLeft,yLeft,xRight,yRight : integer);
begin
  Draw(xLeft,yLeft,xRight,yLeft,3);
  Draw(xRight,yLeft,xRight,yRight,3);
  Draw(xRight,yRight,xLeft,yRight,3);
  Draw(xLeft,yRight,xLeft,yLeft,3);
end;

procedure DrawRings(xCoord,yCoord : integer);
var
  i : byte;
begin
  for i := 1 to 16 do
    Circle(xCoord,yCoord,i * 5,3);
  for i := 1 to 8 do
      FillShape(xCoord + i * 10 - 3,yCoord,3,3);
end;

begin

  GraphMode;
  Palette(1);

  DrawBorder(0,0,319,199);
```

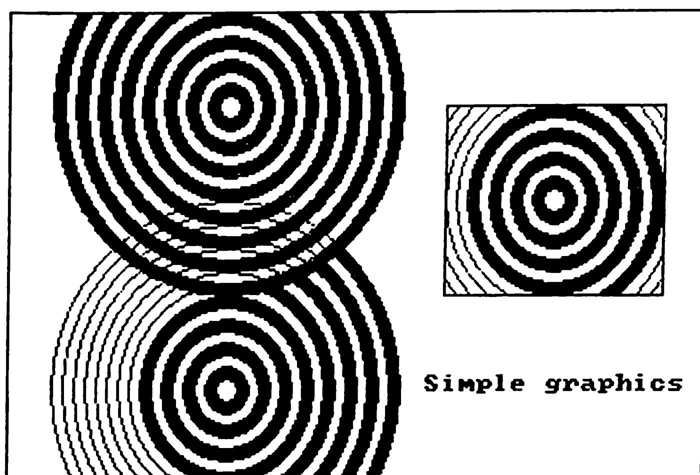
```
    DrawRings(100,40);
    DrawRings(100,160);

    DrawBorder(200,40,300,120);
    GraphWindow(200,40,300,120);
    DrawRings(50,40);

    GotoXY(65,20);
    Write('Simple graphics');

    repeat until KeyPressed;

    TextMode
end.
```



Rys. 21.2 Wykorzystanie prostych podprogramów graficznych

Program *PrintColors—1*

Program ten, przedstawiony na wydruku 21.3, demonstruje wyprowadzanie obszarów w różnych kolorach. Jego rezultatem jest rys. 21.3, na którym poszczególne barwy zostały przedstawione w różnych odcieniach szarości. Ponieważ paletą domniemaną jest paleta nr 0, a tło wybrano jako niebieskie, na monitorze kolorowym pojawią się cztery prostokąty w kolorach niebieskim, zielonym, czerwonym i brązowym. Napisy wyprowadzane za pomocą procedury *Write* będą wyświetlone w kolorze brązowym na niebieskim tle.

Wydruk 21.3 Program *PrintColors-1*

```
program PrintColors;

{$i Graph.p }

var
  x,y,Color : byte;

procedure DrawBorder(xMin,yMIn,xMax,yMax : integer;
                     Color : byte);
begin
  Draw(xMin,yMin,xMax,yMin,Color);
  Draw(xMax,yMin,xMax,yMax,Color);
  Draw(xMax,yMax,xMin,yMax,Color);
  Draw(xMin,yMax,xMin,yMin,Color)
end;

procedure FillWindow(xMin,yMin,xMax,yMax : integer;
                     Color : byte);
begin
  GraphWindow(xMin,yMin,xMax,yMax);
  FillScreen(Color)
end;

begin

  GraphColorMode;
  GraphBackground(1);

  DrawBorder(0,0,319,199,3);

  DrawBorder(10,10,155,95,3);
  DrawBorder(160,10,309,95,3);
  DrawBorder(10,104,155,189,3);
  DrawBorder(160,104,309,189,3);

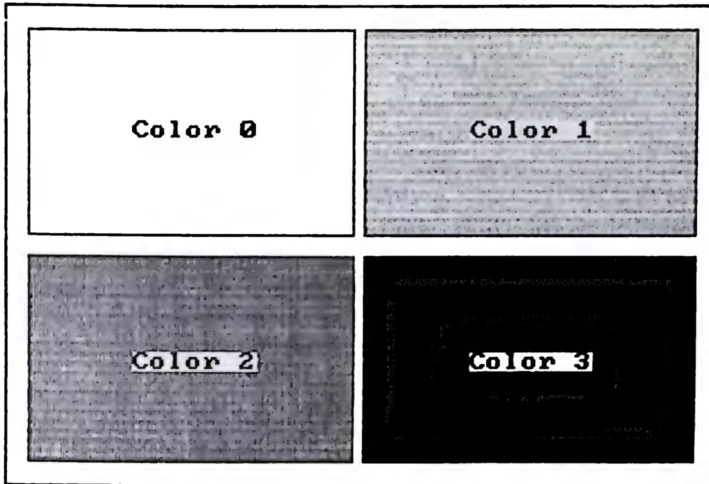
  FillWindow(11,11,154,94,      0);
  FillWindow(161,11,308,94,     1);
  FillWindow(11,105,154,188,    2);
  FillWindow(161,105,308,188,   3);

  Color := -1;
  for y := 0 to 1 do
    for x := 0 to 1 do begin
      Color := Color + 1;
      gotoXY(8 + 19 * x,7 + 12 * y);
      Write('Color',Color :2)
    end;

  repeat until KeyPressed;

  TextMode

end.
```



Rys. 21.3 Wyprowadzanie obszarów w kolorach

Program *PrintColors—2*

Program ten, przedstawiony na wydruku 21.4, stanowi odmianę programu z wydruku 21.3. Demostruje on wyprowadzanie odcinków liniii prostych w różnych kolorach. Jego rezultatem jest rys. 21.4, na którym odcinki w różnych kolorach zostały przedstawione w różnych odcieniach szarości.

Wydruk 21.4 Program *PrintColors-2*

```
program PrintColors;

{$i Graph.p }

var
  x,y,Color : byte;

procedure DrawBorder(xMin,yMin,xMax,yMax : integer;
                     Color : byte);
begin
  Draw(xMin,yMin,xMax,yMin,Color);
  Draw(xMax,yMin,xMax,yMax,Color);
  Draw(xMax,yMax,xMin,yMax,Color);
  Draw(xMin,yMax,xMin,yMin,Color)
end;

procedure FillWindow(xMin,yMin,xMax,yMax : integer;
                     Color : byte);
begin
  GraphWindow(xMin,yMin,xMax,yMax);
  DrawBorder(20,20,xMax - xMin - 20,
                  yMax - yMin - 20,Color);
```

```
  Draw(20,20,xMax - xMin - 20,yMax - yMin - 20,Color);
  Draw(20,yMax - yMin -20,xMax - xMin - 20,20,Color)
end;

begin

  GraphColorMode;
  GraphBackground(0);

  DrawBorder(0,0,319,199,3);

  DrawBorder(10,10,155,95,3);
  DrawBorder(160,10,309,95,3);
  DrawBorder(10,104,155,189,3);
  DrawBorder(160,104,309,189,3);

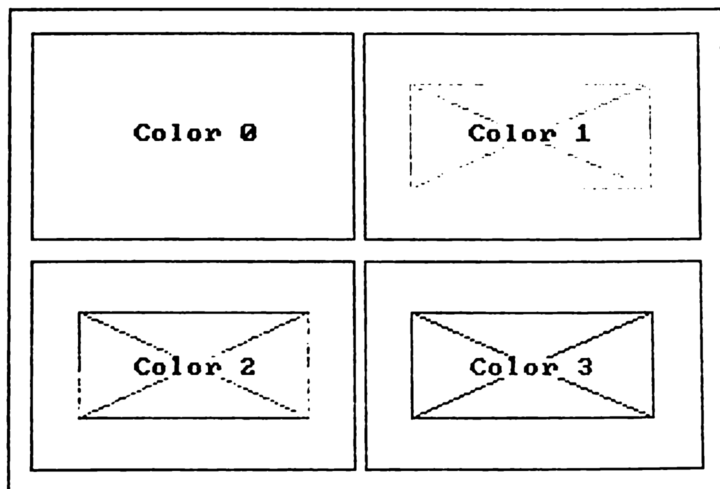
  FillWindow(11,11,154,94,      0);
  FillWindow(161,11,308,94,     1);
  FillWindow(11,105,154,188,    2);
  FillWindow(161,105,308,188,   3);

  Color := -1;
  for y := 0 to 1 do
    for x := 0 to 1 do begin
      Color := Color + 1;
      gotoXY(8 + 19 * x,7 + 12 * y);
      Write('Color',Color :2)
    end;

  repeat until KeyPressed;

  TextMode

end.
```



Rys. 21.4 Wyprowadzanie odcinków linii prostych

Program *ArcAndCircle*;

Program ten, przedstawiony na wydruku 21.5a, demonstruje użycie procedur *Arc* i *Circle*. Ponadto wyjaśnia on zasadę przenoszenia rysunków z ekranu monitora na drukarkę wierszową. Rezultatem wykonania programu jest rys. 21.5a, który w wersji zrealizowanej za pomocą procedury *PrintScreen* ma postać jak na rys. 21.5b. Procedura *PrintScreen* jest zawarta w zbiorze *HardCopy.ibm* i jest przytoczona na wydruku 21.5b. Najistotniejszym elementem proceduryy *PrintScreen* jest funkcja *PixelOn* odwołująca się do obszaru pamięci ekranu. Rezultatem tej funkcji jest dana typu *boolean* określająca prowadziwość zdania „w trybie graficznym wysokiej rozdzielczości punkt ekranu o współrzędnych pikselowych (x,y) świeci w kolorze różnym od koloru tła".

Wydruk 21.5a Program *ArcAndCircle*

```
program ArcAndCircle;

{$i Graph.p }
{$i HardCopy.ibm }

begin

  HiRes;

  Draw(0,0,639,0,1);
  Draw(639,0,639,199,1);
  Draw(639,199,0,199,1);
  Draw(0,199,0,0,1);

  Arc(100,100,270,90,1);
  gotoXY(57,13);
  Write('Arc and Circle');
  Circle(280,100,90,1);

  PrintScreen(^['L');

  repeat until KeyPressed;

  TextMode

end.
```

Wydruk 21.5b Procedura *PrintScreen*

```
type
  ModeString = string[255];

procedure PrintScreen(Mode : ModeString);

  const ScreenBase = $B800;

  var
    x,yL : integer;
    Break : boolean;

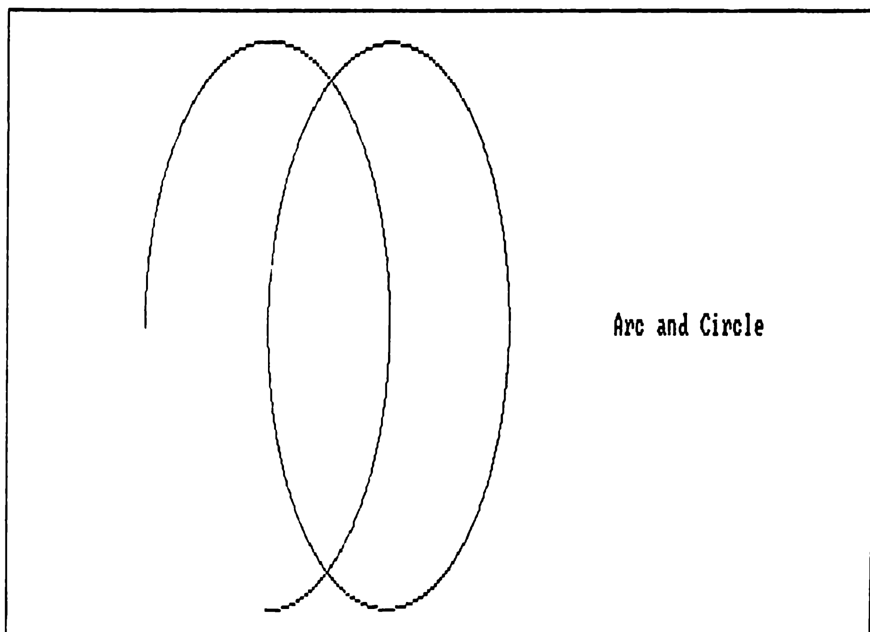
  procedure OneLine;

     function PixelOn(x,y : integer) : boolean;
     begin
       PixelOn := (Mem[ScreenBase :
                     (y and  1) shl 13 +
                     (y and -2) shl  5 +
                     (y and -2) shl  3 + x shr 3] and
                     (128 shr (x and 7)) <> 0)
     end;

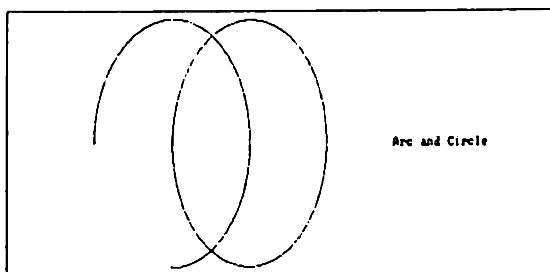
     function OneChar(x,yL : integer) : byte;
     const Bits : array [0..7] of byte =
                  (128,64,32,16,8,4,2,1);
     var OneByte,i : byte;
     begin
       if KeyPressed then break := true;
       yL := yL shl 3;
       OneByte := 0;
       for i := 0 to 7 do
         if PixelOn(x,yL + i) then
           OneByte := OneByte or Bits[i];
       OneChar := OneByte
     end;

   begin  { OneLine }
     if not break then begin
       Write(lst,Mode);
       Write(lst,chr(Lo(640)),chr(Hi(640)));
       for x := 0 to 639 do
         Write(lst,chr(OneChar(x,yL)));
       Writeln(lst)
     end
   end;

 begin  { HardCopy }
   break := false;
   Write(lst,^['@'^['3'#24);
   for yL := 0 to 24 do OneLine;
   Writeln(lst,^['@');
 end;
```



Rys. 21.5a Wyprowadzanie łuków i okręgów



Rys. 21.5b Wyprowadzanie obrazu ekranu

Program *GraphBackground*

Program ten, przedstawiony na wydruku 21.6a, demonstruje posługiwanie się oknami tekstowymi. Rezultatem wykonania programu jest rys. 21.6 zawierający szereg zachodzących na siebie okien tekstowych, w których umieszczono fragment tekstu „To be or not to be".
Ze względu na to, że dostępne programy systemowe do przenoszenia obrazu z ekranu na papier nie umożliwiają traktowania obrazu uzyskanego w trybie znakowym tak, jakby był uzyskany w trybie graficznym, przytoczony rysunek

otrzymano inną drogą, a mianowicie za pomocą wykonania programu *Monolog* przedstawionego na wydruku 21.6b.

Wydruk 21.6a Program *GraphBackground*

```
program GraphBackground;

{$i Graph.p }

var
  Color,i  : byte;

begin

  TextMode(C40);

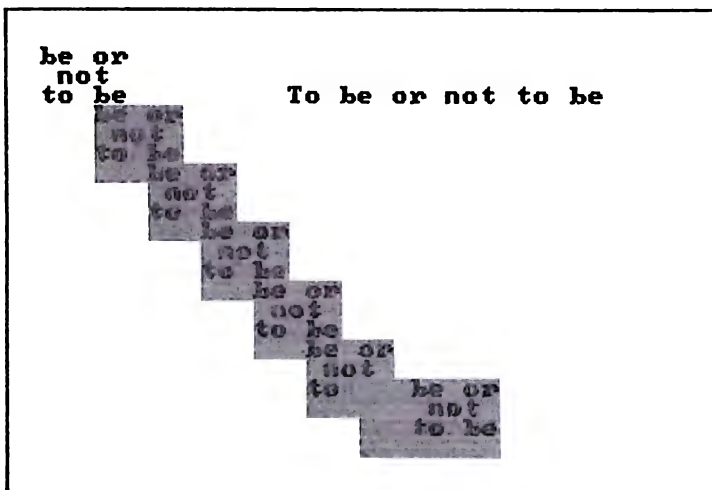
  GraphBackground(3);
  for Color := 0 to 7 do begin
    TextBackground(Color);
    i := 3 * Color + 3;
    Window(i,i,i + 4,i + 3);
    gotoXY(1,1);
    Write('  To be or not to be  ')
  end;

  Window(1,1,40,25);
  gotoXY(15,5);
  TextBackground(0);
  Write('  To be or not to be  ');

  repeat until KeyPressed;

  TextMode(C80);

end.
```



Rys. 21.6 Posługiwanie się oknami tekstowymi

Wydruk 21.6b Program *Monolog*

```
program Monolog;

{$i Graph.p }

var
  Color,Pos : byte;

procedure DrawBorder(xMin,yMIn,xMax,yMax : integer;
                     Color : byte);
begin
  Draw(xMin,yMin,xMax,yMin,Color);
  Draw(xMax,yMin,xMax,yMax,Color);
  Draw(xMax,yMax,xMin,yMax,Color);
  Draw(xMin,yMax,xMin,yMin,Color)
end;

procedure DrawTextWindow(xCor,yCor : integer;Color : byte);
var
  xxCor,yyCor,i : integer;

procedure DrawBackground;
var
  x,y : integer;
begin
  for x := 0 to 39 do
    for y := 0 to 23 do
      if GetDotColor(x,y) <> 3 then
        Plot(x,y,1)
end;

begin { DrawTextWindow }
  if xCor > 35 then xCor := 35;
  if yCor > 20 then yCor := 20;
  xxCor := 8 * (xCor - 1);
  yyCor := 8 * (yCor - 1);
  GraphWindow(xxCor,yyCor,xxCor + 39,yyCor + 31);
  ClearScreen;
  if Color <> 0 then FillScreen(1);
  if Color <> 6 then
    for i := 0 to 2 do begin
      gotoXY(xCor,yCor + i);
      Write(Copy('To be or not to be',5 * i + 4,5))
    end;
  if Color <> 0 then DrawBackground
end;

begin

  GraphColorMode;
  DrawBorder(0,0,319,199,3);
  for Color := 0 to 7 do begin
    Pos := 3 * Color + 3;
    DrawTextWindow(Pos,Pos,Color)
  end;

  gotoXY(15,5);
  GraphBackground(0);
  Write('  To be or not to be  ');

  repeat until KeyPressed;

  TextMode

end.
```

Program *TurtleWindow*

Program ten, przedstawiony na wydruku 21.7, demonstruje posługiwanie się grafiką żółwiową. Rezultatem jego wykonania jest rys. 21.7 zawierający m.in. okno żółwiowe i umieszczone w nim wykresy gwiazdek o przypadkowych rozmiarach. Gwiazdki — w liczbie 20 — są kreślone za pomocą procedury *DrawStar*.

Wydruk 21.7 Program *TurtleWindow*

```
program TurtleWindow;

{$i Graph.p }

var
  Count,x,y : integer;
  Side : integer;

procedure DrawBorder(xMin,yMIn,xMax,yMax : integer);
begin
  Draw(xMin,yMin,xMax,yMin,3);
  Draw(xMax,yMin,xMax,yMax,3);
  Draw(xMax,yMax,xMin,yMax,3);
  Draw(xMin,yMax,xMin,yMin,3)
end;

procedure DrawStar(Side : integer);
var
  i : byte;
begin
  TurnRight(15);
  for i := 1 to 5 do begin
    Forwd(Side);
    TurnRight(78);
    Forwd(Side);
    TurnLeft(150)
  end
end;

begin

  GraphColorMode;
  ClearScreen;
  FillScreen(1);
  DrawBorder(0,0,319,199);

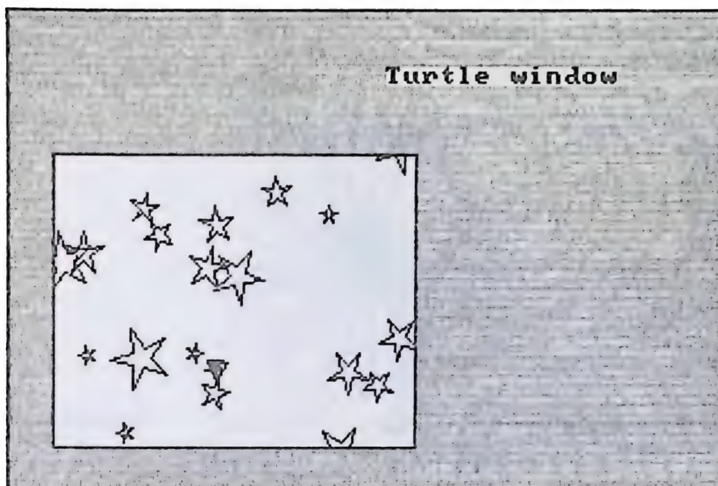
  DrawBorder(20,60,181,181);
  TurtleWindow(100,120,160,120);
  ClearScreen;
  ShowTurtle;

  for Count := 1 to 20 do begin
    x := random(140) - 70;
    y := random(120) - 60;
    Side := random(8) + 3;
    SetPosition(x,y);
    DrawStar(Side)
  end;
  gotoXY(22,4);
  Write('Turtle window');

  repeat until KeyPressed;

  TextMode.

end.
```



Rys. 21.7 Posługiwanie się grafiką żółwiową

Program *TurtleGraphics*

Program ten, przedstawiony na wydruku 21.8, także demonstruje posługiwanie się grafiką żółwiową. Rezultatem jego wykonania jest rys. 21.8 zawierający stokrotkę utworzoną z okręgów i łuków okręgowych. Ponieważ wykreślanie łodygi odbywa się po wywołaniu procedury *Wrap*, ta część łodygi, która wykracza poza dolną krawędź okna pojawia się u góry ekranu.

Wydruk 21.8 Program *TurtleGraphics*

```
program TurtleGraphics;

{$i Graph.p }

procedure aCircle(Rad : integer);
var
  i : integer;
  Side : integer;
begin
  PenUp;
  Forwd(Rad);
  TurnRight(90);
  PenDown;
  Side := trunc(Pi * Rad / 18);
  for i := 1 to 36 do begin
    Forwd(Side);
    TurnRight(10)
  end;
  PenUp;
  TurnLeft(90);
  Back(Rad);
  PenDown
end;
```

```
procedure anArc(Side,Angle : integer);
var
  i : integer;
begin
  TurnRight(5);
  for i := 1 to trunc(Angle / 10) do begin
    Forwd(Side);
    TurnRight(10)
  end;
  TurnRight(-5)
end;

procedure aPetal(Size,Angle : integer);
begin
  anArc(Size,Angle);
  TurnRight(180 - Angle);
  anArc(Size,Angle);
  TurnRight(180 - Angle)
end;

procedure aDaisy(Size,Angle,Count : integer);
var
  i : integer;
begin
  for i := 1 to Count do begin
    aPetal(Size,Angle);
    TurnRight(trunc(360 / Count))
  end;
  Home;
  aCircle(trunc(Size))
end;


begin

  GraphColorMode;

  Draw(0,0,319,0,3);
  Draw(319,0,319,199,3);
  Draw(319,199,0,199,3);
  Draw(0,199,0,0,3);

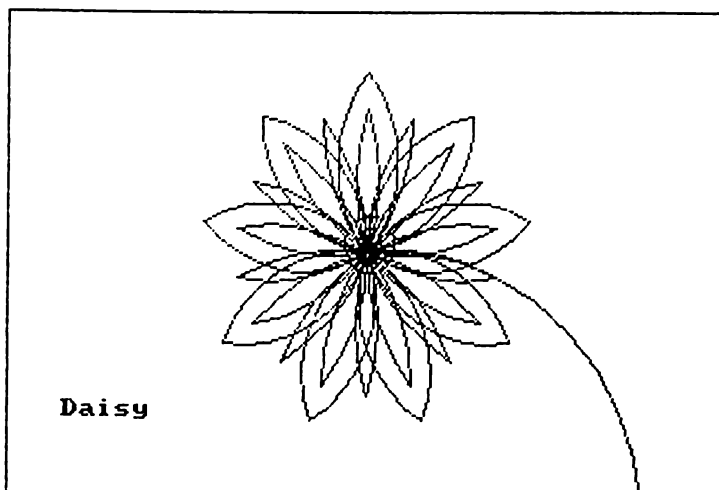
  aDaisy(10,80,9);
  aDaisy(15,40,18);
  Wrap;
  TurnLeft(275);
  anArc(20,90);

  gotoXY(4,21);
  Write('Daisy');

  repeat until KeyPressed;

  TextMode

end.
```



Rys. 21.8 Wykorzystanie podprogramów grafiki żółwiowej

Program *Filling*

Program ten, przedstawiony na wydruku 21.9, demonstruje zasadę wypełniania obszarów. Rezultatem jego wykonania jest rys. 21.9 zawierający parę kwadratów: przed i po dokonaniu wypełniania za pomocą procedury *FillShape*.

Wywołanie procedury *FillShape* zawiera określenie współrzędnych punktu położonego w sąsiedztwie lewego-górnego narożnika prawego kwadratu. Otoczenie tego punktu ma zostać wypełnione kolorem 3 bieżącej palety. Wydawałoby się, że wypełnianie będzie dotyczyć jedynie paska wzdłuż obrzeża kwadratu. Tak jednak nie jest, ponieważ kolor wypełniający obszar przenika do wnętrza kwadratu poprzez dziurki rozmieszczone przypadkowo w kwadracikach siatki prostokątnej.

Wydruk 21.9 Program *Filling*

```
program Filling;

{$i Graph.p }

const
  Size = 4;
  n = 9;
  Hor = 319;
  Ver = 199;

var
  InSide,OutSide : integer;
```

```
  x,y,i,d,dd : integer;
  Seed,Margin,Count : integer;

function Rand(n : integer) : integer ;
begin
  Seed := abs(Seed * 13);
  Rand := (Lo(Seed) xor Hi(Seed))  mod n
end;

procedure ClearDot(x,y : integer);
begin
  if GetDotColor(x,y) <> 0 then begin
    Plot(x,y,0);
    Count := Count - 1
  end
end;

procedure Grid(xCor,yCor : integer);
begin
  Draw(xCor,yCor,xCor + OutSide,yCor,3);
  Draw(xCor + OutSide,yCor,xCor + OutSide,yCor + OutSide,3);
  Draw(xCor + OutSide,yCor + OutSide,xCor,yCor + OutSide,3);
  Draw(xCor,yCor + OutSide,xCor,yCor,3);
  xCor := xCor + d;
  yCor := yCor + d;
  for i := 0 to n do begin
    Draw(xCor,yCor + i * dd,xCor + InSide,yCor + i * dd,3);
    Draw(xCor + i * dd,yCor,xCor + i * dd,yCor + InSide,3)
  end;
  Seed := $AA;
  Count := 3 * n * (n + 1) shr 2;
  repeat .
    x := Rand(n);
    y := Rand(n);
    x := xCor + (2 * x + 1) * d;
    y := yCor + (2 * y + 1) * d;
    case Rand(4) of
      0 : { N } ClearDot(x,y - d);
      1 : { E } ClearDot(x + d,y);
      2 : { S } ClearDot(x,y + d);
      3 : { W } ClearDot(x - d,y)
    end
  until Count = 0
end;


begin

  GraphColorMode;

  Draw(0,0,Hor,0,3);
  Draw(Hor,0,Hor,Ver,3);
  Draw(Hor,Ver,0,Ver,3);
  Draw(0,Ver,0,0,3);

  d := Size + 1;
  dd := 2 * d;
  InSide := n * dd;
  OutSide := Inside + 2 * Size + 2;
  Margin := trunc((Hor + 1 - 2 * OutSide) / 3);
  Grid(Margin,30);
  Grid(2 * Margin + OutSide,30);

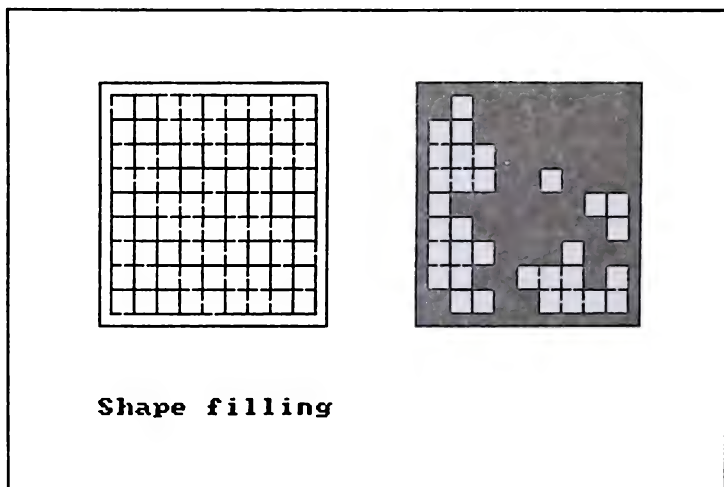
  FillShape(2 * Margin + OutSide + 1,31,3,3);
```

```
  gotoXY(6,21);
  Write('Shape filling');

  repeat until KeyPressed;

  TextMode

end.
```



Rys. 21.9 Wypełnianie obszarów

Program *Logo*

Program ten, przedstawiony na wydruku 21.10, demonstruje umieszczenie na ekranie dowolnego wzoru stanowiącego układ punktów. Rezultatem wykonania programu jest rys. 21.10, zawierający napis jb w otoczce z dwóch prostokątów. Wzór składający się na taki układ punktów jest prostokątem 16×24 pikseli i jest zakodowany w tablicy prostokątnej o nazwie Logo.

Wydruk 21.10 Program *Logo*

```
program Logo;

procedure Logo(xLogo,yLogo : integer);

const
  Logo : array[0..15,0..2] of byte =
          (($FF,$FF,$FF),($80,$00,$01),
           ($BF,$FF,$FD),($A0,$00,$05),
           ($A0,$8E,$05),($A0,$06,$05),
           ($A1,$C7,$C5),($A0,$C6,$65),
           ($A0,$C6,$65),($AC,$C6,$65),
           ($AC,$CB,$C5),($A7,$80,$05),
           ($A0,$00,$05),($BF,$FF,$FD),
           ($80,$00,$01),($FF,$FF,$FF));
var
  OneByte,x,y,i : byte;
```

```
begin
  for y := 0 to 15 do
    for x := 0 to 2 do begin
      OneByte := Logo[y,x];
      for i := 0 to 7 do
        if (OneByte and (128 shr i)) <> 0 then
          Plot(xLogo + x * 8 + i,yLogo + y,1)
    end
end;

procedure jbLogo;
begin
  Logo(610,180)
end;

procedure DrawBorder;
begin
  Draw(0,0,639,0,1);
  Draw(639,0,639,199,1);
  Draw(639,199,0,199,1);
  Draw(0,199,0,0,1);
end;

begin

  HiRes;

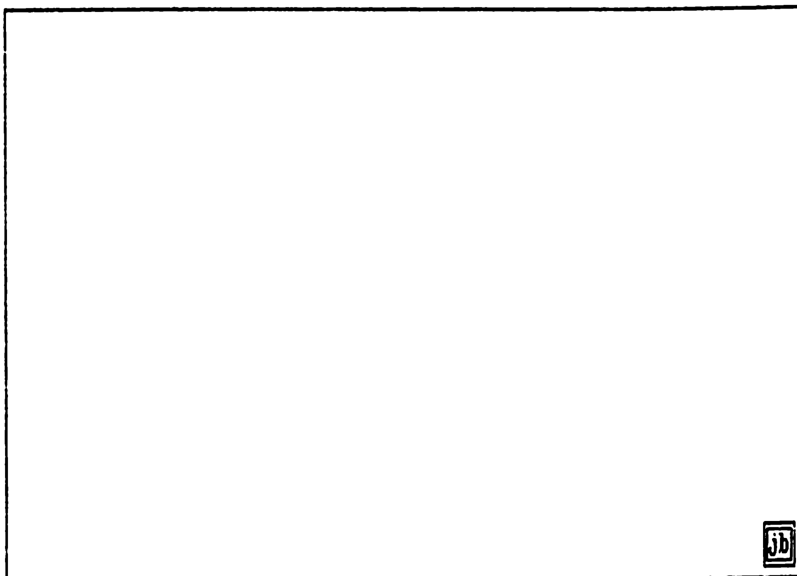
  DrawBorder;

  jbLogo;

  repeat until KeyPressed;

  TextMode

end.
```



Rys. 21.10 Wyprowadzanie zadanego układu punktów

Program *jb*

Program ten, przedstawiony na wydruku 21.11a, demonstruje wypełnienie wybranego obszaru wzorem określonym za pomocą procedury *Pattern*. Ponadto ilustruje on dokonanie włączenia zbioru zawierającego dowolne procedury źródłowe. Rezultatem wykonania programu, korzystającego ze zbioru włączonego, przedstawionego na wydruku 21.11b, jest rys. 21.11. Na rysunku tym widać tło utworzone za pomocą procedury *FillPattern* i nałożony na nie wzór składający się z pary liter jb innego kroju.

Wydruk 21.11a Program *jb*

```
program jb;

{$i Graph.p  }
{$i Logo.sys }

const
  jb : array[0..7] of byte =
           ($02,$35,$54,$54,$34,$10,$14,$00);
  Color = 3;

begin

  GraphColorMode;

  Pattern(jb);
  FillPattern(0,0,319,199,Color);

  jbLogo;

  repeat until KeyPressed;

  TextMode

end.
```

Wydruk 21.11b Procedury włączane przez program *jb*

```
{ Logo.sys }

procedure Logo(xLogo,yLogo : integer);

const
  Logo : array[0..15,0..2] of byte =

           (($FF,$FF,$FF),($80,$00,$01),
            ($BF,$FF,$FD),($A0,$00,$05),
            ($A0,$8E,$05),($A0,$06,$05),
            ($A1,$C7,$C5),($A0,$C6,$65),
            ($A0,$C6,$65),($AC,$C6,$65),
            ($AC,$CF,$C5),($A7,$80,$05),
            ($A0,$00,$05),($BF,$FF,$FD),
            ($80,$00,$01),($FF,$FF,$FF));
var
  OneByte,x,y,i : byte;

begin
  for y := 0 to 15 do
```

```
    for x := 0 to 2 do begin
      OneByte := Logo[y,x];
      for i := 0 to 7 do
        Plot(xLogo + x * 8 + i,yLogo + y,
             ord((OneByte and (128 shr i)) <> 0) shl 1)
    end
end;

procedure jbLogo;
begin
  Logo(290,180)
end;
```

Rys. 21.11 Wypełnianie obszaru zadanym wzorem

Program *PatternDesigner*

Program ten, przedstawiony na wydruku 21.12, demonstruje wypełnianie obszarów wzorami o rozmiarach 8×8 pikseli i służy do interakcyjnego przygotowywania takich wzorów.

Bezpośrednio po rozpoczęciu wykonywania programu na ekranie pojawia się szachownica o rozmiarach 8×8, po której można się dowolnie poruszać, posługując się 8 klawiszami kierunkowymi, w tym 4 klawiszami oznaczonymi strzałkami – w górę, w dół, w lewo i w prawo. Naciśnięcie klawisza Ins powoduje zabarwienie kwadracika szachownicy, a naciśnięcie klawisza Del – jego odbarwienie.

Jednocześnie z wypełnianiem szachownicy zabarwionymi kwadracikami, z jej prawej strony pojawiają się szesnastkowe kody poszczególnych jej elementów,

a pod tymi kodami — wzór reprezentowany na szachownicy, zmniejszony do jednego znaku.

Poza opisanymi klawiszami, można posługiwać się klawiszami sterującymi F1—F4, którym przypisano następujące znaczenie

F1 — Zakończenie wykonywania programu.
F2 — Wypełnienie wzorem dużego prostokąta zajmującego lewą część ekranu.
F3 — Przywrócenie początkowych warunków wykonywania programu.
F4 — Przedstawienie wzoru we wnętrzu okręgu.

Naciśnięcie dowolnego z pozostałych klawiszy nie ma żadnych skutków.

Przykładowe rezultaty interakcyjnej sesji współpracy z programem przedstawiono na rys. 21.12a i 21.12b.

Wydruk 21.12 Program *PatternDesigner*

```
program PatternDesigner;

{$i Graph.p }
{$i Logo.sys }

type
  PatternType = array[0..7] of byte;

const
  xMin =   8; yMin =  14 ;
  xMax = 161; yMax = 159;
  xPad =  25; yPad =   9;
  HexDigits : string[16] = '0123456789ABCDEF';
  Hor = 319;
  Ver = 199;
  EmptyPattern : PatternType =
                    (0,0,0,0,0,0,0,0);
var
  PatternArray,Vector : PatternType;
  Row : byte;
  x,y,xPix,yPix,i,j : integer;
  Ch : char;
  Finish : boolean;

procedure DrawBorder(xMin,yMin,
                     xMax,yMax : integer;
                     Color : byte );
begin
  Draw(xMin,yMin,xMax,yMin,Color);
  Draw(xMax,yMin,xMax,yMax,Color);
  Draw(xMax,yMax,xMin,yMax,Color);
  Draw(xMin,yMax,xMin,yMin,Color)
end;

procedure DrawGrid;
begin
  xPix := (xPad - 1) * 8;
```

```
    yPix := (yPad - 1) * 8;
    for i := 0 to 8 do begin
      Draw(xPix,yPix + 8 * i,
           xPix + 64,yPix + 8 * i,2);
      Draw(xPix + 8 * i,yPix,
           xPix + 8 * i,yPix + 64,2);
      if i < 8 then begin
        gotoXY(xPad + 9,yPad + i);
        Write('00')
      end
    end
  end;

  procedure InvertPixel;
  var
    Pixel : integer;
  begin
    xPix := (xPad - 1) * 8 + x * 8 + 4;
    yPix := (yPad - 1) * 8 + y * 8 + 4;
    Pixel := GetDotColor(xPix,yPix);
    Plot(xPix,yPix,3 - Pixel)
  end;

  procedure ChangePad(Insert : boolean);
  var
    Mask : byte;
  begin
    Mask := 128 shr x;
    Row := PatternArray[y];
    Row := Row and not Mask;
    if Insert then
      Row := Row or Mask;
    PatternArray[y] := Row;
    xPix := (xPad - 1) * 8 + x * 8 + 2;
    yPix := (yPad - 1) * 8 + y * 8 + 2;
    FillShape(xPix,yPix,0,2);
    if Insert then
     FillShape(xPix,yPix,3,2);
    gotoXY(xPad + 9,yPad + y);
    Write(HexDigits[Row shr 4  + 1],
          HexDigits[Row and $F + 1]);
  end;

  procedure DisplayVector;
  begin
    gotoXY(2,23);
    Write('(');
    for j := 0 to 7 do begin
      Row := Vector[j];
      Write(HexDigits[Row shr 4  + 1],
            HexDigits[Row and $F + 1]);
      if j < 7 then Write(',')
    end;
    Write(')')
  end;

  procedure DrawPattern;
  const
    xPix = 266;
    yPix = 148;
  var
    x,y : integer;
  begin
    DrawBorder(xPix,yPix,xPix + 11,yPix + 11,2);
    for y := 0 to 7 do
      for x := 0 to 7 do
```

```
        Plot(xPix + 2 + x,yPix + 2 + y,
             3 * (ord(PatternArray[y] and
                  (128 shr x) <> 0)))
  end;

  procedure FillSquare;
  begin
    GraphWindow(xMin + 1,yMin + 1,
                xMax - 1,yMax - 1);
    ClearScreen;
    GraphWindow(0,0,Hor,Ver);
    for i := 0 to 7 do begin
      Row := PatternArray[7 - i];
      for j := 0 to 7 do begin
        Vector[i] := Vector[i] shl 1 + Row and 1;
        Row := Row shr 1
      end
    end;
    DisplayVector;
    Pattern(Vector);
    FillPattern(xMin + 1,yMin + 1,
                xMax - 1,yMax - 1,3)
  end;

  procedure Trim;
  begin
    Circle((xMin + xMax) shr 1,
           (yMin + yMax) shr 1,
           (xMax - xMin) shr 1 - 10,2);
    FillShape(xMin + 1,yMin + 1,1,2)
  end;

  procedure Initialize;
  begin
    ClearScreen;
    jbLogo;
    DrawBorder(0,0,Hor,Ver,3);
    DrawBorder(xMin,yMin,xMax,yMax,2);
    DrawGrid;
    gotoXY(23,4);
    Write('Pattern Designer');
    x := 0;
    y := 0;
    PatternArray := EmptyPattern;
    InvertPixel
  end;

  begin

    GraphColorMode;

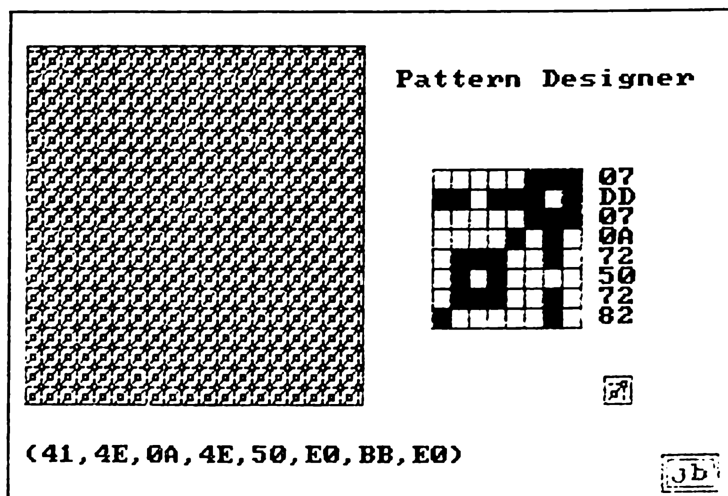
    Initialize;

    Finish := false;
    repeat
      DrawPattern;
      Read(Kbd,Ch);
      if (Ch = ^[) and KeyPressed then begin
        Read(Kbd,Ch);
        InvertPixel;
        case ord(Ch) of
          75 : { W }   x := x - 1;
          77 : { E }   x := x + 1;
          72 : { N }   y := y - 1;
          80 : { S }   y := y + 1;
```

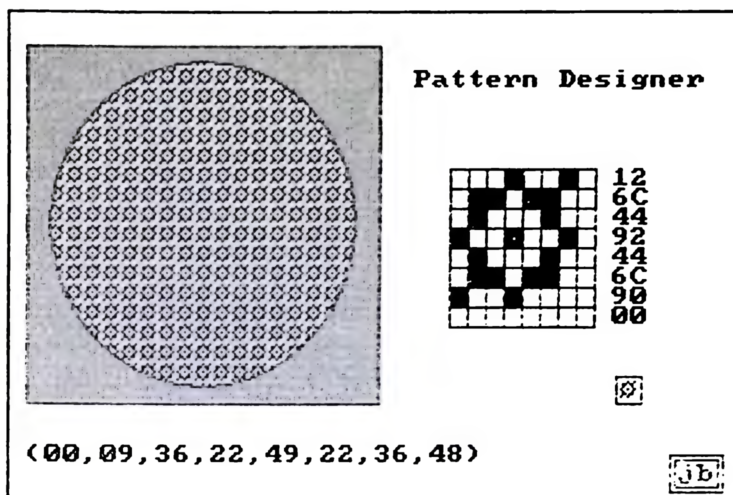
```
            71 : { NW } begin
                          y := y - 1;
                          x := x - 1
                        end;
            73 : { NE } begin
                          y := y - 1;
                          x := x + 1
                        end;
            79 : { SW } begin
                          y := y + 1;
                          x := x -1
                        end;
            81 : { SE } begin
                          y := y + 1;
                          x := x + 1
                        end;
            82 : { Ins } ChangePad(true);
            83 : { Del } ChangePad(false);
            62 : { F4 }  Trim;
            61 : { F3 }  Initialize;
            60 : { F2 }  FillSquare;
            59 : { F1 }  Finish := true
          end;
          x := x and 7;
          y := y and 7;
          if ord(ch) <> 61 then InvertPixel
        end
      until Finish;

      TextMode

    end.
```



Rys. 21.12a Projektowanie wzoru zawartego w kwadracie



Rys. 21.12b Projektowanie wzoru zawartego w okręgu

Program *Surface*

Program ten, przedstawiony na wydruku 21.13, służy do sporządzania wykresów powierzchni trójwymiarowych o równaniu $z = f(x, y)$. Powierzchnia jest przedstawiana w postaci siatki prostokątnej. Wykreśleniu podlega część powierzchni, której rzut na płaszczyznę $z = 0$ zawiera się w przedziale (*xMin,xMax*) i (*yMin,yMax*). Zakłada się, że oko obserwatora znajduje się w punkcie o współrzędnych sferycznych (*Rad,Theta,Phi*) w odległości *D* od ekranu, na który odbywa się rzutowanie.

Wykonanie programu dzieli się na dwie fazy. W pierwszej następuje wyznaczenie ekstremalnych punktów powstałych z rzutowania powierzchni na ekran, a w drugiej utworzenie okna pokrywającego się z niemal całym ekranem i wykreślenie powierzchni. Na rys. 21.13 przedstawiono przykładowy rezultat wykonania programu dla powierzchni

$$z = \cos(\sqrt{x^2 + y^2})$$

Wydruk 21.13 Program *Surface*

```
program Surface;

{$i Graph.p }

const
  x1 = 0;
  y1 = 0;
  x2 = 639;
  y2 = 199;
```

```
var
  xMin,xMax,yMin,yMax : real;
  xCount,yCount : integer;
  x,y,z,dx,dy,Ax,Ay,Bx,By : real;
  dxMin,dxMax,dyMin,dyMax : real;
  xSize,ySize : real;
  xOld,yOld,xNew,yNew : integer;
  xStep,yStep : real;
  Rad,Theta,Phi,D : real;
  i,j : integer;
  Show : boolean;
  sinT,cosT,sinP,cosP : real;

const
  Big = 9.999999E+10;
  Margin = 0.1;

function Fun(x,y : real) : real;
begin
  Fun :=  cos(sqrt(x * x + y * y));
end;

procedure FindEyeCoordinates;
var
  xx,yy,zz : real;
begin
  z := Fun(x,y);
  xx := -x * sinT + y * cosT;
  yy := -x * cosT * cosP - y * sinT * cosP + z * sinP;
  zz := -x * cosT * sinP - y * sinT * sinP - z * cosP + Rad;
  dx := D * xx / zz;
  dy := D * yy / zz
end;

procedure FindScreenCoordinates;
begin
  xNew := trunc(Ax + Bx * dx);
  yNew := y2 - trunc(Ay + By * dy)
end;

procedure FindLimits;
begin
  if dx > dxMax then dxMax := dx;
  if dx < dxMin then dxMin := dx;
  if dy > dyMax then dyMax := dy;
  if dy < dyMin then dyMin := dy
end;




procedure FindWindow;
begin
  xSize := dxMax - dxMin;
  ySize := dyMax - dyMin;
  dxMin := dxMin - Margin * xSize;
  dxMax := dxMax + Margin * xSize;
  dyMin := dyMin - Margin * ySize;
  dyMax := dyMax + Margin * ySize;
  Bx := (x2 - x1) / (dxMax - dxMin);
  By := (y2 - y1) / (dyMax - dyMin);
  Ax := x1 - dxMin * Bx;
  Ay := y1 - dyMin * By
end;
```

```
begin

   xCount := 20;
   yCount := 20;
   xMin := -5;
   xMax :=  5;
   yMin := -5;
   yMax :=  5;
   Rad := 30;
   Theta := -0.5;
   Phi := 0.6;
   D := 10;
   HiRes;
   Draw(0,0,639,0,1);
   Draw(639,0,639,199,1);
   Draw(639,199,0,199,1);
   Draw(0,199,0,0,1);
   sinT := sin(Theta);
   cosT := cos(Theta);
   sinP := sin(Phi);
   cosP := cos(Phi);
   xStep := (xMax - xMin) / xCount;
   yStep := (yMax - yMin) / yCount;
   dxMin := Big;
   dxMax := - Big;
   dyMin := Big;
   dyMax := -Big;

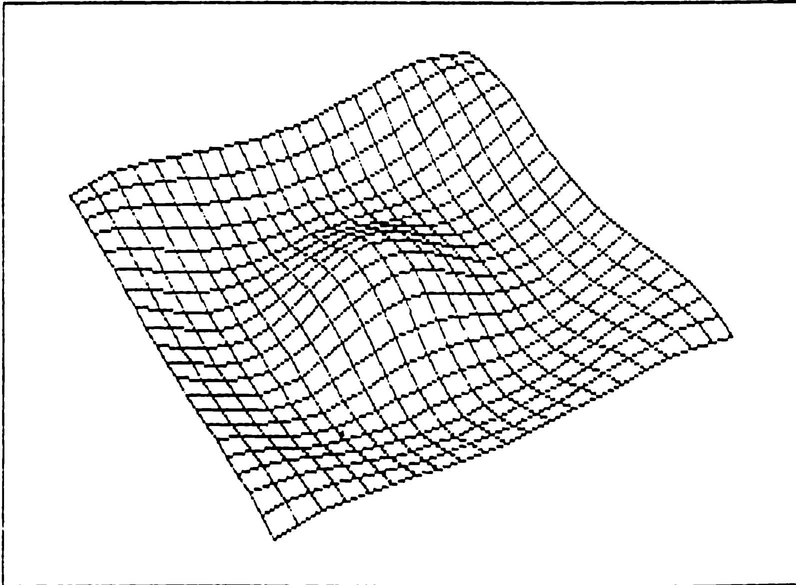

for Show := false to true do begin
  for i := 0 to xCount do begin
    x := xMin + i * xStep;
    y := yMin;
    FindEyeCoordinates;
    if Show then begin
      FindScreenCoordinates;
      xOld := xNew;
      yOld := yNew;
      Plot(xOld,yOld,1)
    end else FindLimits;
    for j := 0 to yCount do begin
      y := yMin + j * yStep;
      FindEyeCoordinates;
      if Show then begin
        FindScreenCoordinates;
        Draw(xOld,yOld,xNew,yNew,1);
        xOld := xNew;
        yOld := yNew
      end else FindLimits
    end
  end;
  for i := 0 to yCount do begin
    y := yMin + i * yStep;
    x := xMin;
    FindEyeCoordinates;
    if Show then begin
      FindScreenCoordinates;
      xOld := xNew;
      yOld := yNew;
      Plot(xOld,yOld,1)
    end else FindLimits;
    for j := 0 to xCount do begin
      x := xMin + j * xStep;
      FindEyeCoordinates;
      if Show then begin
        FindScreenCoordinates;
```

```
            Draw(xOld,yOld,xNew,yNew,1);
            xOld := xNew;
            yOld := yNew
          end else FindLimits
        end
      end;
      if not Show then FindWindow
    end;

    repeat until KeyPressed;

    TextMode

  end.
```



Rys. 21.13 Wykreślanie powierzchni trójwymiarowych

Program *Animation*

Program ten, przedstawiony na wydruku 21.14, demonstruje użycie procedur *GetPic* i *PutPic* dla uzyskania efektów ruchowych. Rezultatem wykonania programu jest postać biegnącego chłopczyka, przedstawiona na rys. 21.14 w różnych fazach ruchu.

Wydruk 21.14 Program *Animation*

```
program Animation;

{$i Graph.p }

const
  Boy : array[0..3] of array[0..3,0..3,0..7] of byte =


        (((($00,$00,$00,$00,$00,$00,$00,$00),
           ($00,$00,$00,$00,$00,$00,$00,$00),
           ($00,$00,$01,$07,$1F,$3F,$3F,$3F),
           ($00,$78,$FC,$F4,$F0,$F0,$F0,$B0)),

          (($00,$00,$00,$00,$00,$00,$00,$00),
           ($00,$00,$01,$03,$07,$0E,$0E,$0C),
           ($3F,$1F,$CF,$E7,$F7,$7F,$7E,$FE),
           ($B0,$F8,$B8,$C8,$F0,$80,$00,$08)),

          (($00,$00,$01,$03,$07,$0F,$0E,$0C),
           ($01,$07,$CF,$CF,$DF,$FF,$FF,$EF),
           ($FE,$FE,$FE,$FF,$F7,$E3,$80,$80),
           ($78,$78,$F0,$C0,$80,$00,$00,$00)),

          (($00,$00,$00,$00,$00,$00,$00,$00),
           ($C3,$03,$03,$01,$01,$00,$00,$00),
           ($80,$83,$87,$CF,$FE,$FC,$F0,$60),
           ($00,$00,$00,$00,$00,$00,$00,$00))),


         ((($00,$00,$00,$00,$00,$00,$00,$00),
           ($00,$00,$00,$00,$00,$00,$00,$00),
           ($00,$00,$00,$00,$00,$00,$03,$0F),
           ($00,$00,$00,$00,$7C,$FE,$FE,$FC)),

          (($00,$00,$00,$00,$00,$00;$00,$00),
           ($00,$00,$00,$00,$01,$03,$07,$06),
           ($1F,$3F;$3F,$1F,$9F,$CF,$E7,$FF),
           ($F8,$F8,$F8,$D8,$D8,$FC,$DC,$E4)),

          (($00,$00,$00,$00,$00,$00,$00,$00),
           ($06,$06,$0F,$3F,$7F,$FF,$FF,$7F),
           ($7F,$FE,$FC,$FC,$FC,$FD,$DF,$8F),
           ($F8,$00,$00,$60,$E0,$E0,$C0,$80)),

          (($00,$00,$01,$01,$00,$00,$00,$00),
           ($7F,$FF,$EF,$CF,$1E,$7F,$FF,$FC),
           ($80,$C0,$C0,$00,$00,$80,$00,$00),
           ($00,$00,$00,$00,$00,$00,$00,$00))),

         ((($00,$00,$00,$00,$00,$00,$00,$00),
           ($00,$00,$00,$00,$00,$00,$00,$00),
           ($00,$00,$00,$03,$07,$0F,$0F,$0F),
           ($00,$00,$7E,$FF,$FF,$FF,$FC,$EC)),

          (($00,$00,$00,$00,$00,$00,$00,$00),
           ($00,$00,$00,$00,$00,$00,$00,$00),
           ($0F,$0F,$07,$01,$01,$07,$3F,$FF),
           ($EC,$FE,$EE,$F2,$FC,$E0,$80,$00)),

          (($00,$00,$00,$00,$00,$00,$00,$00),
           ($01,$01,$03,$07,$0F,$1F,$3F,$FF),
           ($FF,$FF,$FE,$FF,$FF,$FC,$FF,$FF),
           ($00,$00,$00,$80,$80,$00,$80,$00)),
```

```
               (($01,$03,$0F,$1E,$1E,$1C,$1E,$1E),
                ($FC,$C0,$00,$00,$00,$00,$00,$00),
                ($FC,$3E,$0E,$00,$00,$00,$00,$00),
                ($00,$00,$00,$00,$00,$00,$00,$00))),

              ((($00,$00,$00,$00,$00,$00,$00,$00),
                ($00,$00,$00,$00,$00,$00,$00,$00),
                ($00,$01,$07,$0F,$0F,$0F,$0F,$07),
                ($00,$FE,$FF,$FF,$FD,$FC,$EC,$EC)),

               (($00,$00,$00,$00,$00,$00,$3F,$FE),
                ($0D,$1F,$1E,$00,$00,$00,$07,$1F),
                ($C7,$E3,$79,$3F,$3F,$FE,$FF,$FF),
                ($FE,$EE,$F2,$FC,$00,$03,$87,$FF)),

               (($FE,$FC,$3F,$3F,$1F,$00,$00,$00),
                ($3F,$FF,$FF,$FF,$FF,$00,$00,$00),
                ($FF,$F0,$E0,$C0,$F0,$F8,$3F,$1F),
                ($FC,$00,$00,$60,$E0,$E0,$C0,$C0)),

               (($00,$00,$00,$00,$00,$00,$00,$00),
                ($00,$00,$00,$00,$00,$00,$00,$00),
                ($0F,$06,$00,$00,$00,$00,$00,$00),
                ($80,$00,$00,$00,$00,$00,$00,$00))));

  Mask : array[0..7] of byte =
           (128,64,32,16,8,4,2,1);

var
  Buffer : array[0..3,1..166] of byte;
  xCoord,yCoord : integer;
  i,j,k,n,f : byte;
  x,y : integer;
  Ch : char;



begin

  HiRes;

  Draw(0,0,639,0,1);
  Draw(0,0,0,199,1);
  Draw(639,0,639,199,1);
  Draw(0,199,639,199,1);

  for f := 0 to 14 do begin
    xCoord := 16 + f * 40;
    yCoord := 20;
    for i := 0 to 3 do
      for j := 0 to 3 do
        for k := 0 to 7 do begin
          y := yCoord  + i * 8 + k;
          for n := 0 to 7 do begin
            x := xCoord + j * 8 + n;
            if Boy[f mod 4,i,j,k] and Mask[n] <> 0 then
              Plot(x,y,1)
          end
        end;
    if f < 4 then
      GetPic(Buffer[f],xCoord - 8,yCoord,
                       xCoord + 31,yCoord + 31);
  end;
```

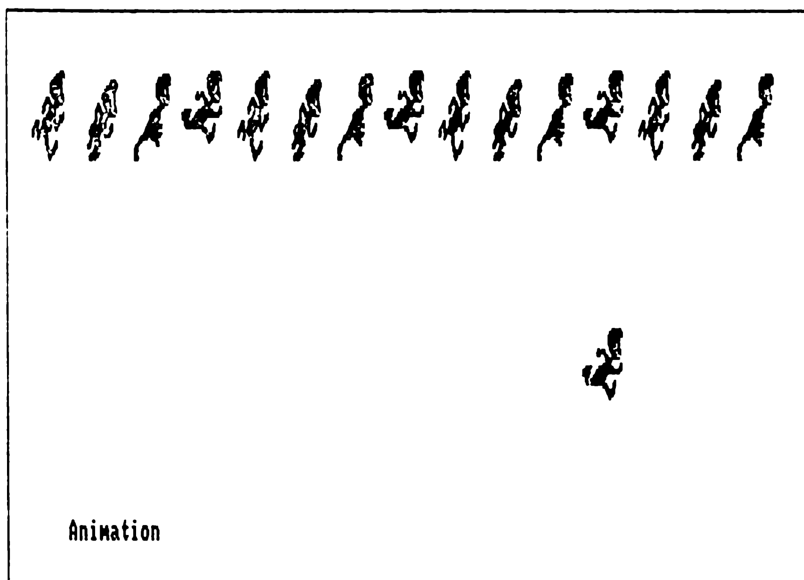
```
  GotoXY(7,23);
  Write('Animation');

  xCoord := 8;
  yCoord := 140;
  for i := 0 to 73 do begin
    if i = 56 then begin
      repeat until KeyPressed;
      Read(Kbd,Ch)
    end;
    PutPic(Buffer[i mod 4],xCoord + i * 8,yCoord);
    Delay(200);
  end;

  repeat until KeyPressed;

  TextMode

end.
```



Rys. 21.14 Uzyskiwanie efektów ruchowych

Dodatek A

Tablica kodu ASCII

dec	hex	char
0	00	^@ NUL
1	01	^A SOH
2	02	^B STX
3	03	^C ETX
4	04	^D EOT
5	05	^E ENQ
6	06	^F ACK
7	07	^G BEL
8	08	^H BS
9	09	^I HT
10	0A	^J LF
11	0B	^K VT
12	0C	^L FF
13	0D	^M CR
14	0E	^N SO
15	0F	^O SI
16	10	^P DLE
17	11	^Q DC1
18	12	^R DC2
19	13	^S DC3
20	14	^T DC4
21	15	^U NAK
22	16	^V SYN
23	17	^W ETB
24	18	^X CAN
25	19	^Y EM
26	1A	^Z SUB
27	1B	^[ESC
28	1C	^\ FS
29	1D	^] GS
30	1E	^^ RS
31	1F	^_ US
32	20	
33	21	!
34	22	”
35	23	#
36	24	$
37	25	%
38	26	&
39	27	’
40	28	(
41	29	)
42	2A	*
43	2B	+
44	2C	,
45	2D	–
46	2E	.
47	2F	/
48	30	0
49	31	1
50	32	2
51	33	3
52	34	4
53	35	5
54	36	6
55	37	7
56	38	8
57	39	9
58	3A	:
59	3B	;
60	3C	<
61	3D	=
62	3E	>
63	3F	?
64	40	@
65	41	A
66	42	B
67	43	C
68	44	D
69	45	E
70	46	F
71	47	G
72	48	H
73	49	I
74	4A	J
75	4B	K
76	4C	L
77	4D	M

<table>
<tr><th>dec</th><th>hex</th><th>char</th></tr>
<tr><td>78</td><td>4E</td><td>N</td></tr>
<tr><td>79</td><td>4F</td><td>O</td></tr>
<tr><td>80</td><td>50</td><td>P</td></tr>
<tr><td>81</td><td>51</td><td>Q</td></tr>
<tr><td>82</td><td>52</td><td>R</td></tr>
<tr><td>83</td><td>53</td><td>S</td></tr>
<tr><td>84</td><td>54</td><td>T</td></tr>
<tr><td>85</td><td>55</td><td>U</td></tr>
<tr><td>86</td><td>56</td><td>V</td></tr>
<tr><td>87</td><td>57</td><td>W</td></tr>
<tr><td>88</td><td>58</td><td>X</td></tr>
<tr><td>89</td><td>59</td><td>Y</td></tr>
<tr><td>90</td><td>5A</td><td>Z</td></tr>
<tr><td>91</td><td>5B</td><td>[</td></tr>
<tr><td>92</td><td>5C</td><td>\</td></tr>
<tr><td>93</td><td>5D</td><td>]</td></tr>
<tr><td>94</td><td>5E</td><td>^</td></tr>
<tr><td>95</td><td>5F</td><td>_</td></tr>
<tr><td>96</td><td>60</td><td>‘</td></tr>
<tr><td>97</td><td>61</td><td>a</td></tr>
<tr><td>98</td><td>62</td><td>b</td></tr>
<tr><td>99</td><td>63</td><td>c</td></tr>
<tr><td>100</td><td>64</td><td>d</td></tr>
<tr><td>101</td><td>65</td><td>e</td></tr>
<tr><td>102</td><td>66</td><td>f</td></tr>
<tr><td>103</td><td>67</td><td>g</td></tr>
<tr><td>104</td><td>68</td><td>h</td></tr>
<tr><td>105</td><td>69</td><td>i</td></tr>
<tr><td>106</td><td>6A</td><td>j</td></tr>
<tr><td>107</td><td>6B</td><td>k</td></tr>
<tr><td>108</td><td>6C</td><td>l</td></tr>
<tr><td>109</td><td>6D</td><td>m</td></tr>
<tr><td>110</td><td>6E</td><td>n</td></tr>
<tr><td>111</td><td>6F</td><td>o</td></tr>
<tr><td>112</td><td>70</td><td>p</td></tr>
<tr><td>113</td><td>71</td><td>q</td></tr>
<tr><td>114</td><td>72</td><td>r</td></tr>
<tr><td>115</td><td>73</td><td>s</td></tr>
<tr><td>116</td><td>74</td><td>t</td></tr>
<tr><td>117</td><td>75</td><td>u</td></tr>
<tr><td>118</td><td>76</td><td>v</td></tr>
<tr><td>119</td><td>77</td><td>w</td></tr>
<tr><td>120</td><td>78</td><td>x</td></tr>
<tr><td>121</td><td>79</td><td>y</td></tr>
<tr><td>122</td><td>7A</td><td>z</td></tr>
<tr><td>123</td><td>7B</td><td>{</td></tr>
<tr><td>124</td><td>7C</td><td>|</td></tr>
<tr><td>125</td><td>7D</td><td>}</td></tr>
<tr><td>126</td><td>7E</td><td>~</td></tr>
<tr><td>127</td><td>7F</td><td>DEL</td></tr>
</table>

Dodatek B

Kody znaków klawiatury IBM PC

Naciśnięcie pewnego znaku klawiatury, ewentualnie łącznie z klawiszem Shift, Ctrl albo Alt, powoduje wprowadzenie do bufora wejściowego jednego albo dwóch znaków. Jeśli zostaną wprowadzone dwa znaki, to pierwszym z nich jest znak Esc o kodzie 27.

Klucze	bez Shift	z Shift	z Ctrl	z Alt
F1	27 59	27 84	27 94	27 104
F2	27 60	27 85	27 95	27 105
F3	27 61	27 86	27 96	27 106
F4	27 62	27 87	27 97	27 107
F5	27 63	27 88	27 98	27 108
F6	27 64	27 89	27 99	27 109
F7	27 65	27 90	27 100	27 110
F8	27 66	27 91	27 101	27 111
F9	27 67	27 92	27 102	27 112
F10	27 68	27 93	27 103	27 113

Klucze specjalne	bez Shift	z Shift	z Ctrl	z Alt
←	27 75	52	27 115	27 178
→	27 77	54	27 116	27 180
↑	27 72	56	27 160	27 175
↓	27 80	50	27 164	27 183
Home	27 71	55		27 174
End	27 79	49	27 117	27 182
PgUp	27 73	57	27 132	27 176
PgDn	27 81	51	27 118	27 184
Ins	27 82	48	27 165	27 185
Del	27 83	46	27 166	27 186
Esc	27	27	27	27
Back	8	8	127	
Tab	9	27 15		
cr	13	13	10	

Litery	bez Shift	z Shift	z Ctrl	z Alt
A	97	65	1	27 30
B	98	66	2	27 48
C	99	67	3	27 46
D	100	68	4	27 32
E	101	69	5	27 18
F	102	70	6	27 33
G	103	71	7	27 34
H	104	72	8	27 35
I	105	73	9	27 23
J	106	74	10	27 36
K	107	75	11	27 37
L	108	76	12	27 38
M	109	77	13	27 50
N	110	78	14	27 49
O	111	79	15	27 24
P	112	80	16	27 25
Q	113	81	17	27 16
R	114	82	18	27 19
S	115	83	19	27 31
T	116	84	20	27 20
U	117	85	21	27 22
V	118	86	22	27 47
W	119	87	23	27 17
X	120	88	24	27 45
Y	121	89	25	27 21
Z	122	90	26	27 44

Pozostałe znaki	bez Shift	z Shift	z Ctrl	z Alt
[	91	123	27	
\	92	124	28	
]	93	125	29	
’	96	126		
0	48	41		27 129
1	49	33		27 120
2	50	64	27 3	27 121
3	51	35		27 122
4	52	36		27 123
5	53	37		27 124
6	54	94	30	27 125
7	55	38		27 126

8	56	42		27 127
9	57	40		27 128
*	42		27 114	
+	43	43		
−	45	95	31	27 130
=	61	43		27 131
,	44	60		
/	47	63		
;	59	58		

Przykład

```
program FunctionKeys;
var
  Ch : char;
begin
  repeat
    Read(Kbd,Ch);
    if (Ch = [ ) and KeyPressed then begin
      Read(Kbd,Ch);
      if ord (Ch) in [59..68] then
        Write ('Key F',ord(Ch)-59)
      else
        Write('Two char key')
    end
    else
     Write('One char key');
    until Ch = ^[
  end.
```

• Jeśli z klawiatury zostanie wyprowadzony jeden ze znaków F1—F10 to zostanie wyprowadzony napis

Key Fn

w którym *n* oznacza numer klucza.

• Jeśli z klawiatury zostanie wprowadzony znak Esc, to spowoduje to zakończenie wykonywania programu.
• W pozostałych przypadkach, stosownie do liczby znaków wprowadzonych do bufora wejściowego, zostanie wyprowadzony napis

One char key

albo

Two char key. □

Dodatek C

Błędy sygnalizowane podczas wykonywania programów

Błędy, które mogą wystąpić podczas wykonywania programów dzielą się na *błędy fatalne* i *błędy operacji wejścia/wyjścia.*

Błędy fatalne są sygnalizowane za pomocą napisu

Run-time error NN, PC = addr
Program aborted

w którym *NN* jest numerem błędu, a *addr* jest adresem tego miejsca w programie, gdzie wykryto błąd.

Błędy operacji wejścia/wyjścia są sygnalizowane za pomocą napisu

I/0 error NN, PC = addr
Program aborted

w którym *NN* i *addr* mają taką samą interpretację jak powyżej.

Błędy fatalne

01	Nadmiar stałopozycyjny.
02	Dzielenie przez zero.
03	Błąd argumentu funkcji *Sqrt.*
04	Błąd argumentu funkcji *Ln.*
10	Próba utworzenia danej łańcuchowej o rozmiarze przekraczającym 255 znaków. Próba konwersji danej łańcuchowej, reprezentującej więcej niż 1 znak, w daną znakową.
90	Niewłaściwy indeks w odwołaniu do tablicy.
91	Próba przypisania danej spoza dopuszczalnego zakresu.
92	Wartość argumentu funkcji *Trunc* albo *Round* spoza dopuszczalnego zakresu.
FF	Brak wolnego miejsca w pamięci.

Błędy operacji wejścia/wyjścia

01	Plik nie istnieje.
02	Plik nie jest otwarty.
03	Plik nie jest otwarty do wyprowadzania.
04	Plik nie jest otwarty.
10	Błąd reprezentacji liczby.
20	Zabroniona operacja.
21	Zabroniona operacja.
22	Zabronione użycie procedury *Assign*.
90	Niezgodność rozmiarów rekordów.
99	Nieoczekiwany koniec pliku.
F0	Błąd operacji zapisu na dysk.
F1	Przepełnienie katalogu.
F2	Przepełnienie pliku.
FF	Brak pliku.

Dodatek D

Edytor ekranowy

Pojęcia podstawowe

Edytor tekstów WordStar został wprowadzony na rynek mikrokomputerowy przez firmę MicroPro już w 1977 roku. Główną zaletą, która w znacznej mierze przyczyniła się do jego rozpowszechnienia, było założenie natychmiastowej edycji ekranowej oraz przyjęcie rozwiązań niezależnych od sprzętu niezbędnego do wdrożenia edytora. Mimo pojawienia się wielu konkurencyjnych implementacji edytorów tekstów, WordStar zachował swoją wiodącą pozycję i w postaci kilku mutacji jest dostępny zarówno na mikrokomputerach 8-bitowych jak i 16-bitowych. W dalszej części opisu zostanie on przedstawiony w postaci, w jakiej jest implementowany w systemie Turbo Pascal.

W odróżnieniu od wielu innych edytorów edytor WordStar jest zorientowany na wprowadzenie tekstu, a nie na przyjmowanie dyrektyw przetwarzania tekstu. Konsekwencją tego założenia jest sterowanie operacjami na tekście za pomocą znaków, które nie mają bezpośredniej reprezentacji graficznej. Takie znaki będą nazywane *znakami sterującymi*, a ich użycie będzie się sprowadzać do jednoczesnego naciśnięcia klawisza znaku CTRL oraz jednego albo dwóch klawiszy znaków literowych dodatkowych. Ponieważ w dalszym opisie odwoływanie się do znaków sterujących będzie występować bardzo często, zostanie przyjęta umowa oznaczania znaków sterujących za pomocą symbolu ^ (caret) reprezentującego naciśnięcie znaku CTRL. W szczególności nadanie znaku sterującego CTRL A będzie w skrócie nazywane wydaniem dyrektywy ^A, co uzyskuje się przez naciśnięcie klawisza CTRL i jednocześnie klawisza oznaczonego literą A. W tych przypadkach gdy sterowanie edytorem odbywa się za pomocą pary znaków sterujących, np. ^KB (także nazywanej dyrektywą) należy nacisnąć klawisz CTRL, a następnie kolejno klawisze oznaczone literami K i B (podczas naciskania drugiego z tych klawiszy zbędne jest wciskanie klawisza CTRL).

16 – Turbo Pascal...

Typowa sesja współpracy z edytorem WordStar zaczyna się od wydania w głównym menu dyrektywy E. Powoduje to wywołanie edytora i umieszczenie na ekranie monitora początkowej części zbioru roboczego zidentyfikowanego uprzednio za pomocą dyrektywy W. Po wprowadzeniu tekstu tego zbioru albo po dokonaniu w nim zmian, należy posłużyć się dyrektywą edytora ^KD. Spowoduje to zakończenie edycji i powrót do głównego menu. W tym momencie można posłużyć się dyrektywą S — dla zapamiętania zbioru roboczego na dysku.

Bezpośrednio po wywołaniu edytora można traktować komputer jako wielofunkcyjną maszynę do pisania. Maszyna taka ma tę właściwość, że każdy wprowadzony znak pojawia się natychmiast na ekranie. Błędnie wprowadzone pojedyncze litery można usuwać za pomocą klawisza (Backspace). Jeśli błędy występują we wcześniej wprowadzonych słowach, to można przesunąć kursor nad takie słowa i dokonać natychmiastowej poprawki. Zasady takiego poprawiania zostaną omówione w następnym punkcie.

Wprowadzenie, zmiany i usuwanie porcji tekstu

Jak już wyjaśniono, bezpośrednio po aktywowaniu edytora, następuje przejście do tworzenia albo aktualizowania tekstu. Odbywa się to poprzez wprowadzanie słów albo operowanie na słowach już wprowadzonych. Pod pojęciem słowa jest rozumiany dowolny ciąg znaków zakończony spacją albo jednym ze znaków przestankowych, takich jak np., (przecinek), . (kropka), : (dwukropek), ' (apostrof).

W obrębie tekstu wyświetlanego na ekranie monitora jeden ze znaków jest wyróżniony za pomocą kursora. Operacje na znakach i słowach są zawsze wykonywane w odniesieniu do tego znaku, zawierającego go słowa albo wiersza. Przesuwanie kursora może także dotyczyć pojedynczych znaków, słów albo wierszy. Może ponadto dotyczyć ekranów, tj. porcji tekstu obejmujących większą liczbę wierszy, zbliżoną do pojemności ekranu monitora.

Przesunięcie kursora o jedną pozycję w lewo, w prawo, w górę i w dół odbywa się za pomocą dyrektyw ^E, ^S, ^D i ^X, których klawisze tworzą romb

```
  E
S   D
  X
```

Jak łatwo się domyślić, dyrektywa ^E przesuwa kursor o jeden wiersz w górę, dyrektywa ^X przesuwa go o jeden wiersz w dół, a dyrektywy ^S i ^D przesuwają kursor odpowiednio o jedną pozycję w lewo i o jedną pozycję w prawo. Dzięki takiemu rozwiązaniu, o funkcji dyrektywy decyduje nie

mnemonika nazwy jej klawisza lecz położenie klawisza w przytoczonym układzie czterokierunkowym.

Przesuwanie kursora o pojedyncze słowa jest realizowane za pomocą dwóch dodatkowych dyrektyw ^A i ^F

```
      E
A S     D F
      X
```

Użycie dyrektywy ^A powoduje przesunięcie kursora o jedno słowo w lewo, a użycie dyrektywy ^F powoduje przesunięcie kursora o jedno słowo w prawo.

Kolejna para dyrektyw: ^R i ^C służy do przesunięcia kursora odpowiednio do poprzedniego i następnego ekranu. Klawisze tych dyrektyw są usytuowane z prawej strony osi wyznaczonej przez klawisze E-X. Z lewej strony tej osi znajduje się para klawiszy, które przesuwają tekst na ekranie o jeden wiersz: ^W w dół i ^Z w górę. Tak więc ostatecznie układ klawiszy wykorzystywanych do sterowania położeniem kursora oraz określających porcję wyświetlanego tekstu przybiera postać

```
W     E     R
A S     D F
Z     X     C
```

Układ ten jest łatwy do zapamiętania i nie wymaga identyfikowania klawiszy z opatrującymi je napisami.

Z prawej strony tego zestawu klawiszy znajdują się trzy klawisze dyrektyw ^T, ^Y i ^G tworzące układ

```
T   Y
  G
```

Użycie dyrektywy ^G powoduje usunięcie znaku wyróżnionego przez kursor, użycie dyrektywy ^T powoduje usunięcie znaków od znaku wyróżnionego przez kursor — do końca słowa, a użycie dyrektywy ^Y powoduje usunięcie wiersza, w którym znajduje się kursor.

Uzupełnieniem przytoczonego repertuaru dyrektyw jest znak (Backspace), którego naciśnięcie powoduje usunięcie znaku poprzedzającego znak wyróżniony przez kursor.

Podsumowanie

^E Przesunięcie kursora o jeden wiersz do góry, z zachowaniem pozycji kolumny.

^X Przesunięcie kursora o jeden wiersz do dołu, z zachowaniem pozycji kolumny.

^S Przesunięcie kursora o jedną pozycję w lewo.
^D Przesunięcie kursora o jedną pozycję w prawo.
^A Przesunięcie kursora do początku słowa w lewo.
^F Przesunięcie kursora do początku słowa w prawo.
^R Przesunięcie „okna", przez które widziany jest dokument o jeden ekran do góry.
^C Przesunięcie „okna", przez które widziany jest dokument o jeden ekran do dołu.
^W Przesunięcie tekstu na ekranie o jeden wiersz w dół.
^Z Przesunięcie tekstu na ekranie o jeden wiersz do góry.
^G Usunięcie znaku wyróżnionego przez kursor.
^T Usunięcie znaku wyróżnionego przez kursor oraz następujących po nim znaków danego słowa.
^Y Usunięcie wiersza, w którym znajduje się znak wyróżniony przez kursor.
Del Usunięcie znaku wyróżnionego przez kursor.

Przytoczony tu repertuar dyrektyw, umożliwiający wizualizację i usuwanie dowolnych partii tekstu został w edytorze WordStar wzbogacony dyrektywą ^N do wstawiania pustych wierszy oraz dodatkowym zestawem dyrektyw realizujących funkcje zbliżone do poprzednio omówionych — tyle że z większym „skokiem". Każda z takich dyrektyw składa się ze znaku ^Q, bezpośrednio po którym występuje jeden ze znaków tworzących układ

```
  E  R
S  D
  X  C
```

Dyrektywa ^QE powoduje przesunięcie kursora do pierwszego wiersza ekranu, dyrektywa ^QX powoduje przesunięcie kursora do ostatniego wiersza ekranu, dyrektywa ^QS powoduje przesunięcie kursora do początku, a dyrektywa ^QD powoduje przesunięcie kursora do końca tego wiersza, w którym znajduje się kursor. Dwie pozostałe dyrektywy ^QR i ^QC powodują odpowiednio przesunięcie kursora do początku i do końca tekstu.

Przesuwanie kursora za pomocą dowolnych zestawów wymienionych tu dyrektyw może służyć dwóm celom: przemieszczaniu się w obrębie tekstu oraz przemieszczaniu się do obszaru, w którym mają być dokonane zmiany. W tym drugim przypadku zmiany mogą polegać na uzupełnieniu tekstu albo zastąpieniu go innym. W celu uzupełnienia tekstu wystarczy wprowadzić go z klawiatury. Edytor wstawi taki tekst przed znak wyróżniony przez kursor, dokonując na bieżąco przesunięcia znaku wyróżnionego oraz znaków po nim następujących. W celu zmiany tekstu można go najpierw usunąć, albo zastąpić nowym — na zasadzie uzupełnienia, albo stary tekst wyrugować nowym. To

ostatnie rozwiązanie nie należy do najlepszych, gdyż rzadko się zdarza aby stary i nowy tekst były tej samej długości. Ponadto jego zastosowanie wymaga przejściowego wyłączenia opcji rozsuwania tekstu, pociągając za sobą konieczność posłużenia się dyrektywą ^V. Każdorazowe użycie tego znaku powoduje przełączenie edytora ze stanu zastępowania tekstu w stan wstawiania tekstu i odwrotnie. Jego dokładne działanie najlepiej jest wypraktykować, śledząc zmiany dokonujące się na ekranie.

Złożone operacje na tekście

Opisane dotychczas dyrektywy umożliwiają nie tylko tworzenie, ale również praktycznie dowolne przetwarzanie dokumentów. Tym niemniej w pewnych przypadkach przydatne są operacje dotyczące większych fragmentów tekstu, takie jak np. przestawienie lub kopiowanie grup sąsiadujących wierszy albo wyszukiwanie lub zmiana określanych słów lub fraz. Do takich celów służą omówione tu dyrektywy blokowe.

Dyrektywa ^QF

Po wykonaniu tej dyrektywy na ekranie pojawia się napis

FIND:

stanowiący zapytanie o frazę, która ma być wyszukana w tekście. Fraza taka składa się z dowolnej porcji znaków zakończonej znakiem Enter. Bezpośrednio po naciśnięciu klawisza Enter na ekranie pojawia się zapytanie

OPTIONS:

na które należy odpowiedzieć określeniem trybu poszukiwania fazy. W najczęściej używanym przypadku — poszukiwania frazy począwszy od pozycji wyróżnionej przez kursor w kierunku końca tekstu, wystarczy w odpowiedzi na zapytanie nacisnąć klawisz Enter. Jeśli poszukiwana fraza znajduje się w tekście, to kursor zostanie umieszczony bezpośrednio za ostatnim znakiem znalezionej frazy. Jak łatwo się domyślić, użycie dyrektywy ^QF nie jest niczym innym jak posłużeniem się dyrektywą edytora kontekstowego. Należy nadmienić, że wśród opcji określających tryb poszukiwania frazy znajdują się m.in. takie jak

B poszukuj wstecz,
W ogranicz się do pełnych słów,
U utożsamiaj litery duże i małe.

Ponadto można posłużyć się opcją mającą postać liczby. W takim przypadku, dla liczby *n*, poszukiwanie dotyczy *n*-tego wystąpienia poszukiwanej frazy.

Dyrektywa ^QA

Dyrektywa ^QA stanowi rozwinięcie dyrektywy ^QF. Bezpośrednio po jej użyciu także pojawia się zapytanie o poszukiwaną frazę, ale po naciśnięciu kończącego ją klawisza Enter pojawia się zapytanie o frazę

REPLACE WITH:

na które należy odpowiedzieć frazą, która ma zastąpić frazę podaną po FIND. Po tej drugiej frazie, także zakończonej naciśnięciem klawisza Enter, pojawia się znane zapytanie o opcje określające tryb poszukiwania frazy. Poza opcjami już wymienionymi dopuszczalne są tu także opcje

G zastępuj w całym tekście (a więc nie od pozycji wyróżnionej przez kursor, lecz od pierwszego znaku tekstu),
N zastępuj bezwarunkowo.

Jeśli użyto opcji N, to wykonanie dyrektywy przebiega aż do wykonania wszystkich zastąpień. Jeśli nie użyto tej opcji, to po każdym znalezieniu poszukiwanej frazy wyświetlane jest zapytanie, czy należy dokonać zastąpienia. Odpowiedź Y stanowi zgodę na zastąpienie. Dowolna inna odpowiedź powoduje zaniechanie proponowanego zastąpienia i podjęcie poszukiwania następnego wystąpienia zastępowanej frazy. Przerwanie tego procesu można uzyskać za pomocą dyrektywy ^U.

Dyrektywa ^KB

Wykonanie dyrektywy ^KB powoduje oznaczenie miejsca w tekście jako tzw. początku bloku. Zdefiniowanie bloku wymaga dodatkowo użycia dyrektywy ^KK.

Dyrektywa ^KK

Wykonanie dyrektywy ^KK powoduje oznaczenie miejsca w tekście jako tzw. końca bloku. Dyrektywa ta wraz z dyrektywa ^KB definiuje blok. Z chwilą zdefiniowania bloku ciąg znaków tekstu znajdujący się pomiędzy pozycjami kursora w chwili wykonywania dyrektyw ^KB i ^KK zostaje wyróżniony za pomocą inwersji znaków albo ich przyciemnienia. Dalej opisane dyrektywy grupy ^Kx będą dotyczyć tej właśnie partii tekstu.

Dyrektywa ^KC

Wykonanie dyrektywy ^KC powoduje skopiowanie bloku wyróżnionego przez dyrektywy ^KB i ^KK na pozycję przed znak wyróżniony przez kursor. Po tej operacji kursor zostanie ustawiony na pierwszym znaku skopiowanego bloku.

Dyrektywa ^KV

Wykonanie dyrektywy ^KV powoduje przeniesienie bloku wyróżnionego przez dyrektywy ^KB i ^KK na pozycję przed znak wyróżniony przez kursor. Po tej operacji kursor zostanie ustawiony na pierwszym znaku przeniesionego bloku, który w swojej nowej pozycji pozostanie blokiem wyróżnionym.

Dyrektywa ^KY

Wykonanie dyrektywy ^KY powoduje usunięcie wyróżnionego bloku. Można dla przykładu nadmienić, że jedynym skutkiem wykonania ciągu dyrektyw ^KB, ^KK, ^KY jest anulowanie wyróżnienia bloku.

Dyrektywa ^KR

Wykonanie dyrektywy ^KR powoduje wstawienie w miejscu wyróżnionym przez kursor tekstu znajdującego się w zbiorze dyskowym. Bezpośrednio po nadaniu tej dyrektywy na ekranie monitora pojawia się zapytanie

Read block from file:

o nazwę włączonego zbioru.

Dyrektywa ^KW

Wykonanie dyrektywy ^KW powoduje wyprowadzenie do zbioru dyskowego bloku tekstu wyróżnionego przez dyrektywy ^KB i ^KK. Bezpośrednio po użyciu tej dyrektywy na ekranie monitora pojawia się zapytanie

Write block from file:

o nazwę zbioru, w którym zostanie umieszczony wyróżniony blok. Jeśli zbiór taki już istnieje, to przed umieszczeniem w nim bloku zostanie usunięty, a następnie utworzony ponownie.

Dyrektywy pomocnicze

Dyrektywa ^N

Wykonanie dyrektywy ^N powoduje wstawienie w miejscu wyróżnionym przez kursor pary znaków cr-lf, tj. wstawienie nowego wiersza.

Dyrektywa ^I

Wykonanie dyrektywy ^I powoduje przesunięcie kursora do najbliższej pozycji tabulacyjnej. Pozycją tą jest początek słowa w wierszu ponad kursorem.

Dyrektywa ^L

Wykonanie dyrektywy ^L powoduje powtórzenie ostatniej dyrektywy ^QF.

Dyrektywa ^U

Wykonanie dyrektywy ^U powoduje zaniechanie wykonania dowolnej dyrektywy.

Dyrektywa ^P

Wykonanie dyrektywy ^P powoduje, że następujący bezpośrednio za nią znak sterujący jest umieszczany w tekście. W szczególności wprowadzenie sekwencji znaków ^P^M^P^J umożliwia identyfikację pary znaków cr-lf kończącej każdy wiersz tekstu.

Przykład
Przesunięcie całego tekstu o 5 kolumn w prawo wymaga wprowadzenia z klawiatury następujących znaków (znak Enter oznaczono symbolem *e*, a spację symbolem *s*)

^QR*sssss*

^QA^P^M^P^J*e*^P^M^P^J*sssssse*NG*r* □

Dodatek E

Grafika w Turbo Pascalu 4.0

Pod koniec 1987 r. pojawiła się na rynku informatycznym nowa wersja języka Turbo Pascal opracowana wyłącznie z przeznaczeniem dla mikrokomputerów rodziny IBM PC. Wersja ta spotkała się z wielkim zainteresowaniem użytkowników, od dawna oczekujących implementacji, w której miały być przełamane liczne ograniczenia i niedogodności wersji 3.0.

Jak wynika z pierwszych ocen, nowa wersja języka spełniła pokładane w niej nadzieje. Do najbardziej znaczących, nowych jej możliwości należy zaliczyć:

- udostępnienie bardzo wygodnego i znacznie rozszerzonego środowiska operacyjnego;
- blisko trzykrotne zwiększenie szybkości kompilowania programów (do 27 tys. wierszy na minutę w przypadku użycia mikrokomputera IBM PC/AT 8MHz);
- umożliwienie kompilowania programów, których kod rezydujący w pamięci operacyjnej zajmuje więcej niż 64 KB;
- udostępnienie środków kompilacji warunkowej i sterowania sposobem generowania kodu za pomocą dyrektyw preprocesora;
- stworzenie możliwości dzielenia dużych programów na rozłącznie kompilowane moduły;
- udostępnienie nowych typów standardowych, ułatwiających programowanie systemowe;
- udostępnienie obszernej, wstępnie skompilowanej biblioteki podprogramów graficznych;
- udostępnienie mechanizmów umożliwiających tworzenie programów wynikowych z podprogramów napisanych w różnych językach programowania;
- zwiększenie jakości generowanego kodu przez jego optymalizację i eliminowanie z programu wynikowego zbędnych podprogramów bibliotecznych;
- zwiększenie zgodności języka ze standardem ANSI.

Bez wątpienia największą nowością Turbo Pascala 4.0 jest koncepcja modułów. Umożliwia ona tworzenie wstępnie skompilowanych bibliotek typów, zmiennych i podprogramów i korzystanie z nich w różnych programach bez potrzeby powtórnej kompilacji. Jedną z takich bibliotek jest moduł *Graph* zawierający ok. 70 nowych podprogramów graficznych, ułatwiających pisanie programów niezależnych od użytego środowiska graficznego.

Struktura modulu nie odbiega istotnie od struktury programu. Podobnie jak program, moduł składa się z nagłówka, deklaracji i części wykonawczej zakończonej kropką. W odróżnieniu od programu, część deklaracyjna modułu jest rozbita na część publiczną i prywatną. Część publiczna charakteryzuje się tym, że wszystkie zadeklarowane w niej obiekty są znane poza modułem. Natomiast obiekty zadeklarowane w części prywatnej są znane tylko w obrębie modułu. Z podziału tego wynika jedna bardzo ważna właściwość modułów, a mianowicie: jeśli moduł *A* korzysta z zasobów modułu *B*, to po skompilowaniu najpierw modułu *B*, a potem modułu *A* i ewentualnym dokonaniu zmian w części prywatnej modułu *B*, nie trzeba już kompilować modułu *A*. O implementacji mającej tę właściwość mówi się, że zapewnia kompilację rozłączną.

Przykład. Program odwołujący się do modułów

```
program Main;
uses
  Lib,Crt;
var
  Fix : byte;
begin
  ClrScr;
  Fix := 6;
  Sub(Fix);
  Write(Fix)
end.
```

```
unit Lib;
interface
  procedure Sub(var Par : byte);
implementation
  procedure Sub;
  begin
    Par := Par+7
  end;
end.
```

gram *Main* zawiera *wyszczególnienie modułów*

```
uses
  Lib,Crt;
```

którym zadeklarowano zamiar posłużenia się przytoczonym wyżej modułem *Lib* oraz modułem bibliotecznym *Crt* (zawartym w bibliotece modułów *TURBO.TPL*). Moduł *Lib* składa się z części publicznej

```
interface
  procedure Sub(var Par : byte);
```

w której podano zewnętrzne właściwości procedury *Sub*, oraz z części prywatnej

```
implementation
  procedure Sub;
  begin
    Par := Par + 7
  end;
end.
```

zawierającej szczegóły implementacyjne tej procedury.

Innym, ważnym rozszerzeniem Turbo Pascala 4.0 są dodatkowe typy całkowite: *shortint*, *longint* i *word*, typy rzeczywiste: *single*, *double*, *extended* i *comp*, oraz typ wskazujący *pointer*.

Typ całkowity *shortint* jest związany ze zbiorem danych o wartościach z przedziału −128..127, typ *longint* jest związany ze zbiorem danych o wartościach z przedziału

−2 147 483 648..2 147 483 647

a typ *word* jest związany ze zbiorem danych o wartościach z przedziału 0..65525. Jak łatwo wywnioskować, dane typu *shortint* są 1-bajtowe, dane typu *word* są 2-bajtowe, a dane typu *longint* są 4-bajtowe.

Typy rzeczywiste *single*, *double*, *extended* i *comp* znacznie ułatwiają i przyspieszają wykonywanie działań na danych zmiennopozycyjnych. Dane tych typów mogą być jednak użyte tylko w programach posługujących się koprocesorem arytmetycznym.

Typ wskazujący *pointer* jest związany ze zbiorem *wskazań adresowych*, to jest takich wskazań, które mogą lokalizować dowolne obszary pamięci operacyjnej, a nie jak dotychczas, tylko takie obszary, w których znajdują się zmienne określonego typu.

Oprócz wyposażenia w moduły i nowe typy danych, Turbo Pascal rozszerzono o środki do deklarowania podprogramów do obsługiwania przerwań i definiowania podprogramów otwartych, wzbogacono go o kilka nowych operatorów oraz rozbudowano o znacznie rozszerzoną, wstępnie skompilowaną bibliotekę podprogramów graficznych. Opis tej biblioteki, zawartej w modułach *Graph* i *Crt*, stanowi istotę niniejszego Dodatku. Zostanie on poprzedzony prezentacją nowego środowiska operacyjnego systemu Turbo Pascal oraz krótkim omówieniem zasad posługiwania się podprogramami graficznymi wersji 4.0.

Środowisko operacyjne

Wywołanie systemu Turbo Pascal odbywa się za pomocą dyrektywy *Turbo*. Bezpośrednio po wykonaniu tej czynności, na ekranie pojawia się główne menu, a na jego tle informacja o numerze wersji systemu. Naciśnięcie dowolnego klawisza klawiatury powoduje usunięcie tej informacji i odsłonięcie ekranu podzielonego na następujące części:
— wiersz menu, składający się z pól File, Edit, Run, Compile i Options,
— okienko edytora, zatytułowane Edit,
— okienko wyjściowe, zatytułowane Output,
— wiersz opisów kluczy, zawierający informacje na temat funkcji przypisanych wybranym klawiszom Fn.

W każdej chwili posługiwania się systemem Turbo Pascal jest aktywny wiersz menu albo jedno z wymienionych okienek. Ponadto na ekran mogą być przywoływane okienka pomocnicze związane z realizowaniem funkcji systemu. W szczególności naciśnięcie w dowolnej chwili klawisza Alt-F10 powoduje wyświetlenie na ekranie okienka pomocniczego zawierającego informację o numerze wersji systemu.

Aktywność wiersza menu jest uwidoczniona przez wyróżnienie przez inwersję jednego z jego pól. Aktywność okienka edycyjnego albo wyjściowego objawia się zwiększeniem jasności słowa tytułującego okienko (odpowiednio Edit albo Output) oraz zmianą pojedynczej linii, na której znajduje się to słowo, na linię podwójną.

Uaktywnienie wiersza menu następuje po naciśnięciu klawisza F10. Czynność ta może być wykonana w dowolnym momencie działania systemu. Wybranie funkcji określonej przez główne menu może być dokonane na kilka sposobów. Najprostszym jest wprowadzenie z klawiatury pierwszej litery pola wymienionego w menu. Można też posłużyć się klawiszami strzałek poziomych, a po przemieszczeniu się na wybrane pole menu, nacisnąć klawisz Enter. W każdym z opisanych przypadków (z wyjątkiem Run i Edit) nastąpi wywołanie podmenu zawierającego opcje. Wybranie funkcji określonej przez opcję podmenu odbywa się analogicznie do wyboru funkcji z menu (tym razem używa się klawiszy strzałek pionowych) i może powodować wywołanie kolejnego podmenu. Należy dodać, że jeśli po wybraniu pierwszego podmenu zostaną użyte klawisze strzałek poziomych, to na ekran będą przywołane sąsiednie podmenu głównego menu. Właściwość ta jest bardzo przydatna w początkowym okresie zapoznawania się z systemem, kiedy rozmieszczenie jego funkcji między podmenu nie jest jeszcze dobrze znane.

W przypadku omyłkowego wyboru podmenu może cofnąć się na wyższy

poziom menu naciskając klawisz Esc. Naciśnięcie tego klawisza w głównym menu powoduje uaktywnienie ostatnio aktywnego okienka edycyjnego albo wyjściowego.

Wybieranie funkcji głównego menu może być znacznie przyspieszone za pomocą klawisza Alt. Naciśnięcie go w dowolnym momencie wraz z pierwszą literą pola głównego menu powoduje natychmiastowe wykonanie funkcji związanej z tym polem. W szczególności oznacza to, że w celu wykonania funkcji Run głównego menu, wystarczy wprowadzić z klawiatury znak Alt-R (zamiast naciskać najpierw klawisz F10, a następnie R).

Oprócz wymienionych klawiszy F10, Alt-F10 i Alt-*p* (gdzie *p* jest pierwszą literą pola głównego menu), wiele użytecznych funkcji spełniają klawisze Fn i Alt-Fn:

F1 — przywołanie na ekran informacji pomocniczych związanych z bieżącym kontekstem użycia systemu;

F2 — zapamiętanie na dysku tekstu znajdującego się w okienku edycyjnym;

F3 — podjęcie akcji umożliwiającej załadowanie do okienka edycyjnego zbioru o dowolnie wybranej nazwie (przez domniemanie przyjmuje się, że jest to zbiór z rozszerzeniem .PAS);

F5 — powiększenie (lub zmniejszenie) okienka aktywnego;

F6 — przełączenie aktywności między okienkiem edycyjnym i wyjściowym; z dowolnego menu — przywrócenie aktywności okienka;

F9 — skompilowanie programu wykonywalnego w trybie *Make* (przez domniemanie przyjmuje się, że chodzi o program źródłowy znajdujący się w okienku edycyjnym);

Alt-F1 — przywołanie na ekran ostatnio wyświetlanych informacji pomocniczych;

Alt-F3 — podjęcie akcji umożliwiającej załadowanie do okienka edycyjnego jednego z ostatnio używanych zbiorów źródłowych;

Alt-F5 — ujawnienie pełnego ekranu, na który są wyprowadzane wyniki wykonywania programów (część tego ekranu jest wyświetlana w okienku wyjściowym);

Alt-F9 — skompilowanie programu albo modułu źródłowego znajdującego się w okienku wyjściowym.

Funkcja określona przez każdy z wymienionych klawiszy może być wykonana w dowolnym kontekście (oczywiście poza okresem wykonywania programu). W celu zakończenia współpracy z systemem Turbo Pascal można posłużyć się klawiszem Alt-X. Spowoduje to wykonanie czynności kończących i wywołanie systemu DOS.

Przygotowywanie programów

Integralną częścią systemu Turbo Pascal jest edytor ekranowy wzorowany na edytorze WordStar. Jest on wywoływany z głównego menu (np. przez wciśnięcie klawisza E) albo z dowolnego innego kontekstu (np. klawisz Alt-E). Tym, którzy nie znają edytora WordStar wystarczy podać, że przemieszczanie kursora w górę, w dół, w lewo i w prawo odbywa się za pomocą klawiszy strzałkowych, usuwanie znaków — za pomocą klawiszy Backspace (←) i Delete (Del), usuwanie wierszy — za pomocą klawisza Ctrl-Y, a wstawienie wierszy pustych — za pomocą klawisza Ctrl-N. Poza wymienionymi, wiele użytecznych funkcji spełniają następujące klawisze:

Home — przesunięcie kursora na początek wiersza;
End — przesunięcie kursora na koniec wiersza;
Ctrl-Home — przesunięcie kursora do pierwszego wiersza ekranu;
Crtl-End — przesunięcie kursora do ostatniego wiersza ekranu;
PgUp — przywołanie na ekran poprzedniej strony;
PgDn — przywołanie na ekran następnej strony;
Ctrl-PgUp — przesunięcie kursora do początku tekstu;
Ctrl-PgDn — przesunięcie kursora do końca tekstu.

Jeśli podczas przygotowywania tekstu źródłowego (programu albo modułu) zostanie naciśnięty klawisz Ctrl-F1, to na ekranie zostaną wyświetlone informacje na temat wyróżnionego przez kursor elementu języka. W szczególności, jeśli będzie to dotyczyć słowa *InitGraph*, które jest nazwą jednej z funkcji graficznych, to na ekranie pojawi się krótki opis tej funkcji oraz wykaz funkcji jej pokrewnych.

Zakończenie redagowania nie wymaga wykonania specjalnych czynności. Pożądane jest jednak zapamiętanie wprowadzonego tekstu w zbiorze dyskowym. W tym celu wystarczy nacisnąć klawisz F2 lub też po wywołaniu głównego menu (klawisz F10) wybrać pole File (klawisz F), a następnie opcję S (Save), albo W (Write).

Zalecanym sposobem przygotowywania zbiorów źródłowych jest przyspieszone wybranie pola File (klawisz Alt-F), a następnie opcji Load (klawisz L). Po wykonaniu tych czynności na ekranie jest wyświetlane okienko pomocnicze, w którym należy podać nazwę zbioru. Jeśli nie jest to nazwa już wyświetlana, to należy wprowadzić ją z klawiatury i nacisnąć klawisz Enter. Jeśli jest zbliżona do wyświetlanej, to można poddać ją redagowaniu i w dowolnym momencie nacisnąć klawisz Enter. Spowoduje to załadowanie podanego zbioru do okienka edycyjnego. Jeśli zbiór taki nie istnieje, to okienko edycyjne zostanie wyczyszczone. Wprowadzony tekst można zapamiętać na dysku (klawisz F2) jako zbiór o uprzednio podanej nazwie.

Innym sposobem przygotowania tekstu jest przyspieszone wybranie pola File (klawisz Alt-F), a następnie wybranie opcji New (klawisz N). Spowoduje to wyczyszczenie okienka edycyjnego i obranie nazwy zbioru NONAME.PAS. Jeśli po zakończeniu redagowania podejmie się akcję zapamiętania tekstu, to na ekran zostanie przywołane okienko zawierające zapytanie, czy nazwa NONAME.PAS nie powinna zostać zmieniona na inną. Po udzieleniu odpowiedzi nastąpi dokończenie akcji.

Przykłady. Tworzenie i modyfikowanie programów.
a. Utworzenie nowego programu i zapamiętanie go na dysku

```
begin
  Write('Hello, I am JanB')
end.
```

Wykaz czynności:

- wybranie pola głównego menu (klawisz Alt-F);
- wybranie opcji Load (klawisz L);
- określenie nazwy zbioru;
- wprowadzenie tekstu programu;
- zapamiętanie zbioru na dysku (klawisz F2).

b. Utworzenie programu bez zapamiętywania go na dysku

Wykaz czynności:

- wybranie pola File głównego menu (klawisz Alt-F);
- wybranie opcji New (klawisz N);
- wprowadzenie tekstu programu.

Wprowadzony program może być poddawany modyfikacjom, kompilacjom i wykonywaniu. Ponieważ jest przechowywany w pamięci operacyjnej, więc nie musi być zamiętany na dysku.

Kompilowanie programów i modułów

Kompilowanie programów i modułów może odbywać się w trybie *Compile*, *Make* i *Build*. Jeśli po wybraniu pola Compile (klawisz Alt-C), zostanie wybrana opcja Destination Disk (klawisz D), to produkt kompilacji zostanie umieszczony w pamięci zewnętrznej. Jeśli kompilacja dotyczy zbioru *Nazwa*.PAS zawierającego program, to zostanie utworzony zbiór *Nazwa*.EXE, a jeśli dotyczy zbioru zawierającego moduł, to zostanie utworzony zbiór *Nazwa*.TPU.

Kompilowaniu w trybie *Compile* podlega program albo moduł załadowany do okienka edycyjnego. W celu wykonania kompilacji w tym trybie, należy

nacisnąć klawisz Alt-F9 (albo kolejno klawisze F10,C,C). Kompilacja w trybie *Compile* jest możliwa tylko wówczas, gdy wszystkie moduły wymienione w wyszczególnieniu modułów są już skompilowane i znajdują się na dysku (albo w bibliotece TURBO.TPL).

Wywołanie kompilatora w trybie *Make* i *Build* różni się od jego wywołania w trybie *Compile* tym, że rezultatem kompilacji jest zawsze program wykonywalny (umieszczony w pamięci operacyjnej albo na dysku, stosownie do stanu opcji Destination).

Wywołanie kompilatora w trybie *Make* powoduje rozpoczęcie kompilacji od programu znajdującego się w *zbiorze pierwotnym*. Jeśli nazwa tego zbioru nie została określona (co można uczynić posługując się klawiszami Alt-C,P), to rozpoczyna się od programu znajdującego się w okienku edycyjnym. Jeśli program odwołuje się bezpośrednio albo pośrednio do modułów, które podlegały modyfikacji już po utworzeniu wykorzystujących je innych modułów, to kompilowane są również i te moduły. W każdym przypadku system ogranicza się jednak tylko do tych kompilacji, które są niezbędne. W szczególności, nie kompiluje się ponownie modułu odwołującego się do innego modułu, w którym wprowadzano zmiany nie naruszające jego części publicznej.

Wywołanie kompilatora w trybie *Build* różni się od jego wywołania w trybie *Make* tym, że są przeprowadzane wszystkie kompilacje, niezależnie od tego czy są niezbędne.

Należy dodać, że nawet gdy program wykonywalny jest umieszczany w pamięci operacyjnej, to skompilowane moduły, zawarte w zbiorach z rozszerzeniem .TPU, są umieszczane w pamięci zewnętrznej.

Przykłady. Kompilowanie programów i modułów

a. Skompilowanie programu znajdującego się w okienku edycyjnym

```
begin
  Write('Hello world')
end.
```

Wykaz czynności:

- naciśnięcie klawisza Alt-F9.

Jeśli kompilacja zakończy się sukcesem, to program można wykonać, naciskając w tym celu klawisz Alt-R.

b. Skompilowanie programu znajdującego się w zbiorze o arbitralnie wybranej nazwie FIRST.PAS

Wykaz czynności:

- określenie nazwy zbioru pierwotnego (klawisze Alt-C, P a następnie nazwa FIRST.PAS);
- wykonanie kompilacji w trybie Make (klawisz F9).

Jeśli kompilacja zakończy się sukcesem, to program można wykonać naciskając klawisz Alt-R.

Usuwanie błędów i wykonywanie programów

Rzadko się zdarza, aby kompilowany program był bezbłędny. Ponieważ kompilator wykrywa wszelkie odstępstwa od wymagań definicji języka, samą kompilację można traktować jako element uruchomienia programu.

Jeśli podczas wykonywania kompilacji wykryto błąd, to kompilacja jest przerywana, a w pierwszym wierszu okienka edycyjnego pojawia się komunikat diagnostyczny. Jednocześnie kursor zostaje usytuowany w pobliżu miejsca wystąpienia błędu. Po poprawieniu błędu, można ponowić kompilację programu albo modułu. Ze względu na dużą szybkość kompilacji można zaakceptować decyzję implementacyjną wykrywania tylko jednego, a nie wszystkich błędów.

Przykład. Program z błędem

```
begin
  Write(Hello world')
end.
```

Skompilowanie przytoczonego programu powoduje wyprowadzenie komunikatu

Error 3: Unknown identifier

(nieznany identyfikator). Oczywiście przyczyna błędu jest inna i polega na tym, że słowo Hello nie jest poprzedzone apostrofem.

Po usunięciu wszystkich błędów wykrywanych przez kompilator, można spowodować wykonanie programu. W tym celu wystarczy wybrać w głównym menu pole Run (np. Alt-R). Jeśli przed wybraniem tego pola nie wykonano kompilacji, to zostanie domniemana kompilacja w trybie *Make* (jak po naciśnięciu klawisza F9).

Przebieg wykonywania programu zależy od tego, czy wpada on w pętlę, czy też kończy się w sposób normalny lub po wystąpieniu błędu fatalnego. Jeśli wpada w pętlę, to może być przerwany za pomocą klawisza Ctrl-Break, ale tylko wówczas, gdy są spełnione dwa warunki: po pierwsze klawisz ten nie został programowo dezaktywowany, a po drugie program wykonuje operacje wejścia-wyjścia. W przeciwnym razie jest konieczne ponowne załadowanie systemu.

Po wykryciu błędu fatalnego na ekranie jest wyświetlany komunikat

Runtime error *nnn* at *ssss*:*oooo*

określający numer błędu (*nnn*) i adres wystąpienia błędu (*ssss*:*oooo*).

Przykład. Program zawierający błąd fatalny

```
begin
  Write(2 / 0)
end.
```

Wykonanie programu zostanie przerwane z powodu dzielenia przez 0. Na ekranie pojawi się wówczas komunikat

Runtime error 200 at 0000:001E

a po naciśnięciu dowolnego klawisza klawiatury, w pierwszym wierszu okienka edycyjnego zostanie wyprowadzony komunikat

Error 200: Division by zero

(błąd nr 200: dzielenie przez zero).

Nawigowanie wśród opcji menu

Bezpośrednio po wywołaniu systemu Turbo Pascal i usunięciu z ekranu informacji o numerze wersji systemu, a także bezpośrednio po naciśnięciu klawisza F10, następuje uaktywnienie głównego menu, składającego się z pól: File, Edit, Run, Compile i Options. Wybranie dowolnego z nich (z wyjątkiem Edit i Run) powoduje uaktywnienie podmenu składającego się z opcji. Znaczenie poszczególnych pól i opcji jest szczegółowo opisane w okienkach pomocniczych wyświetlanych na ekranie po naciśnięciu klawisza F1. Dlatego podane dalej opisy będą ograniczone do minimum.

Pole File

Wybranie pola File powoduje pojawienie się na ekranie następujących opcji:

Load
Pick
New
Save
Write to
Directory
Change dir
OS shell
Quit

Wybranie opcji Load (klawisz L) umożliwia załadowanie zbioru w celu poddania go redagowaniu. Jeśli nazwa zbioru jest niejednoznaczna (zawiera

znaki * albo lub ?), to naciśnięcie klawisza Enter powoduje wyświetlenie zestawu nazw zbiorów. Wybór jednej z nazw odbywa się za pomocą klawiszy oznaczonych strzałkami i klawisza Enter. Jeśli nazwa jest jednoznaczna, to do okienka edycyjnego jest ładowana zawartość zbioru o podanej nazwie.

Wybranie opcji Pick (klawisz P) umożliwia załadowanie do okienka edycyjnego zawartości zbioru wymienionego na *liście wyboru*. Lista wyboru składa· się z co najwyżej 8 pozycji i zawiera nazwy tych zbiorów, które były ostatnio ładowane do okienka edycyjnego. Na liście wyboru znajduje się pozycja "--loadfile--". Wybranie jej ma taki sam skutek jak wybranie opcji Load.

Wybranie opcji New (klawisz N) powoduje wyczyszczenie okienka edycyjnego i przyjęcie przez domniemanie, że zbiór załadowany do okienka ma nazwę NONAME.PAS.

Wybranie opcji Save (klawisz S) powoduje zapamiętanie na dysku, zbioru znajdującego się w okienku edycyjnym. Jeśli zbiór ten ma nazwę NONAME.PAS, to system proponuje zmienić ją na inną.

Wybranie opcji Write to (klawisz W) powoduje zapamiętanie na dysku zbioru znajdującego się w okienku edycyjnym, ale pod inną nazwą. Jeśli podana nazwa pokrywa się z nazwą zbioru już istniejącego, to system upewnia się, czy zbiór ten ma zostać usunięty.

Wybranie opcji Directory (klawisz D) powoduje wyprowadzenie w okienku pomocniczym nazw zbiorów o podanej masce. System proponuje maskę *.*, ale maska ta może być zmieniona na dowolną inną.

Wybranie opcji Change dir (klawisz C) powoduje ujawnienie nazwy bieżącego podkatalogu i umożliwia jej zmianę na inną.

Wybranie opcji OS shell (klawisz D) powoduje tymczasowe wywołanie systemu DOS. Po zakończeniu operacji w systemie DOS można wrócić do systemu Turbo Pascal wykonując dyrektywę Exit.

Wybranie opcji Quit (klawisz Q) powoduje powrót do systemu DOS.

Pole Edit

Wybranie pola Edit powoduje wywołanie edytora i uaktywnienie okienka edycyjnego.

Pole Run

Wybranie pola Run powoduje wykonanie uprzednio skompilowanego programu. Jeśli nie wykonano kompilacji, to jest ona wykonywana niejawnie w trybie Make.

Pole Compile

Wybranie pola Compile powoduje pojawienie się na ekranie następujących opcji:

Compile
Make
Build
Destination
Find error
Primary file
Get info

Wybranie opcji Compile (klawisz C) powoduje wykonanie kompilacji w trybie Compile.

Wybranie opcji Make (klawisz M) powoduje wykonanie kompilacji w trybie Make.

Wybranie opcji Build (klawisz B) powoduje wykonanie kompilacji w trybie Build.

Wybranie opcji Destination (klawisz D) powoduje przełączenie z kompilowania do pamięci operacyjnej (Memory) na kompilowanie do pamięci dyskowej (Disk) i odwrotnie.

Wybranie opcji Find error (klawisz F) powoduje zapytanie o adres błędu, w celu zlokalizowania instrukcji źródłowej zawierającej ten błąd (użycie omawianej opcji jest niezbędne jedynie wówczas, gdy program był wykonywany poza systemem Turbo Pascal).

Wybranie opcji Primary (klawisz P) umożliwia określenie nazwy zbioru pierwotnego.

Wybranie opcji Get info (klawisz G) powoduje udostępnienie obszernych informacji o programie.

Pole Options

Wybranie pola Options powoduje pojawienie się na ekranie następujących opcji:

Compiler
Environment
Directories
Parameters
Load Options
Save Options

Opcja Compiler

Wybranie opcji Compiler powoduje pojawienie się na ekranie następujących podopcji:

Range checking
Stack checking
I/O checking
Debug information
Turbo pascal map file
Force for calls
Var-string checking
Boolean evaluation
Numeric processing
Link buffer
Conditional defines
Memory size

Wybranie podopcji Range checking (klawisz R) powoduje włączenie (On) albo wyłączenie (Off) kontroli indeksów tablic i zmiennych łańcuchowych, a także kontroli poprawności przypisań danych.

Wybranie podopcji Stack checking (klawisz S) powoduje włączenie (On) albo wyłączenie (Off) kontroli przekroczenia pojemności stosu procesora.

Wybranie podopcji I/O checking (klawisz I) powoduje włączenie (On) albo wyłączenie (Off) kontroli poprawności wykonywania operacji wejścia/wyjścia.

Wybranie podopcji Debug information (klawisz D) powoduje włączenie (On) albo wyłączenie (Off) generowania informacji umożliwiającej lokalizowanie błędów wykonywania programu bezpośrednio w jego tekście źródłowym.

Wybranie podopcji Turbo pascal map file (klawisz T) powoduje włączenie (On) albo wyłączenie (Off) generowania zbioru z rozszerzeniem .TPM umożliwiającego uruchomienie programu wynikowego za pomocą takich programów jak *Symdeb* i *Periscope*.

Wybranie podopcji Force far calls (klawisz F) powoduje włączenie (On) albo wyłączenie (Off) takiego sposobu kompilowania podprogramów, aby ich wywołania i powroty były traktowane jako dalekie.

Wybranie podopcji Var-string checking (klawisz V) powoduje włączenie (Strict) albo wyłączenie (Relaxed) kontroli zgodności parametrów i argumentów łańcuchowych.

Wybranie podopcji Boolean evaluation (klawisz B) powoduje włączenie (Complete) albo wyłączenie (ShortCircuit) kompilowania całych wyrażeń logicznych, niezależnie od tego, czy jest to konieczne do określenia ich wartości.

Wybranie podopcji Numeric processing (klawisz N) powoduje włączenie (Hardware) albo wyłączenie (Software) korzystania z koprocesora arytmetycznego i tym samym wykonywanie operacji na danych typu *real* sprzętowo albo programowo.

Wybranie podopcji Link buffer (klawisz L) powoduje włączenie (Memory) albo wyłączenie (Disk) przechowywania wyników pośrednich konsolidacji w pamięci operacyjnej.

Wybranie podopcji Conditional defines (klawisz C) umożliwia definiowanie (oddzielonych średnikami) symboli warunkowych preprocesora.

Wybranie podopcji Memory sizes (klawisz M) umożliwia określenie rozmiaru stosu oraz minimalnego i maksymalnego rozmiaru sterty.

Każda z wymienionych podopcji może być jawnie użyta jako dyrektywa. Jeśli podczas redagowania tekstu źródłowego zostanie naciśnięty klawisz Ctrl-F7, to aktualne podopcje przetworzone na dyrektywy zostaną umieszczone na początku przygotowywanego tekstu.

Opcja Environment

Wybranie opcji Environment powoduje pojawienie się na ekranie następujących podopcji:

Backup source files
Edit auto save
Config auto save
Retain saved screen
Tab size
Zoom windows
Screen size

Wybranie podopcji Backup source files (klawisz B) powoduje włączenie (On) albo wyłączenie (Off) tworzenia zbiorów z rozszerzeniem .BAK, jako ochrony przed utratą poprzednich wersji zbiorów źródłowych.

Wybranie podopcji Edit auto save (klawisz E) powoduje włączenie (On) albo wyłączenie (Off) automatycznego zapamiętywania tekstu znajdującego się w okienku edycyjnym przed wykonaniem programu (np. Alt-R) albo przed wywołaniem systemu DOS (np. Alt-F,O).

Wybranie podopcji Config auto save (klawisz C) powoduje włączenie (On) albo wyłączenie (Off) automatycznego zapamiętania w zbiorze dyskowym, w chwili kończenia pracy pod nadzorem systemu Turbo Pascal (np. Alt-X), wszystkich ustawionych podopcji systemu (w tym podopcji Config auto save).

Wybranie podopcji Retain saved screen (klawisz R) powoduje włączenie (On) albo wyłączenie (Off) przechowywania w pamięci operacyjnej obrazu znajdującego się na ekranie wyjściowym.

Wybranie podopcji Tab size (klawisz T) umożliwia wybranie skoku tabulacji.

Wybranie podopcji Zoom windows (klawisz Z) powoduje włączenie (On) albo wyłączenie (Off) wyświetlania tylko jednego okienka: edycyjnego albo wyjściowego. Wyświetlenie drugiego okienka wymaga wówczas przełączeń między okienkami (klawisz F6).

Wybranie podopcji Screen size (klawisz S) umożliwia określenie pionowego rozmiaru ekranu niestandardowego (43 wiersze dla karty EGA i 50 wierszy dla karty VGA).

Opcja Directories

Wybranie opcji Directories powoduje pojawienie się na ekranie następujących podopcji:

Turbo directory
Executable directory
Include directories
Unit directories
Object directories
Pick file name
Current pick file

Wybranie podopcji Turbo directory (klawisz T) umożliwia określenie nazwy podkatalogu, w którym znajduje się zbiór konfiguracyjny (przez domniemanie TURBO.TP) i zbiór zawierający informacje pomocnicze (TURBO.HLP).

Wybranie podopcji Executable directory (klawisz E) umożliwia określenie nazwy podkatalogu, w którym są umieszczane zbiory z rozszerzeniem .EXE (przez domniemanie przyjmuje się podkatalog bieżący).

Wybranie podopcji Include directories (klawisz I) umożliwia określenie nazw podkatalogów, w których znajdują się zbiory włączane do programów i modułów źródłowych za pomocą dyrektyw {$I *nazwa*}. Nazwy katalogów są oddzielane średnikami. Jeśli lista nazw jest pusta, to przez domniemanie przyjmuje się nazwę podkatalogu bieżącego.

Wybranie podopcji Unit directories (klawisz U) umożliwia określenie nazw podkatalogów, w których są umieszczane zbiory z rozszerzeniem .TPU powstałe ze skompilowania modułów. Nazwy podkatalogów są oddzielane średnikami. Jeśli wykaz nazw jest pusty, to przyjmuje się nazwę podkatalogu bieżącego.

Wybranie podopcji Object directories (klawisz O) umożliwia określenie nazw podkatalogów, w których znajdują się zbiory z rozszerzeniem .OBJ (skompilowane podprogramy asemblerowe). Nazwy podkatalogów są oddzielone średnikami. Jeśli wykaz nazw jest pusty, to przyjmuje się nazwę podkatalogu bieżącego.

Wybranie podopcji Pick file name (klawisz P) umożliwia określenie nazwy zbioru, w którym jest zapamiętywana lista nazw udostępniana po wybraniu opcji Pick (np. Alt-F, P). Jeśli nazwa zbioru nie jest określona, to przyjmuje się nazwę zawartą w podopcji Current pick file.

Wybranie podopcji Current pick file nie jest możliwe. Jest ona wyświetlana tylko w celach informacyjnych.

Opcja Parameters

Wybranie opcji Parameters powoduje pojawienie się na ekranie okienka, umożliwiającego określenie oddzielonych spacjami parametrów programu. Dzięki omawianej opcji można w systemie Turbo Pascal badać zachowanie się programu wywoływanego następnie (z parametrami) z systemu operacyjnego DOS.

Opcja Load Options

Wybranej opcji Load Options umożliwia odtworzenie podopcji, obowiązujących w systemie Turbo Pascal w chwili wybrania takiej opcji Save Options, w której użyto identycznej nazwy jak w Load Options. Przez domniemanie przyjmuje się nazwę TURBO.TP.

Opcja Save Options

Wybranie opcji Save Options umożliwia zapamiętanie aktualnych podopcji w zbiorze o wybranej nazwie.

Moduły *Graph* I *Crt*

Podprogramy zawarte w modułach *Graph* i *Crt* umożliwiają rozpoznawanie środowiska graficznego (np. *Graph*), inicjowanie grafiki pikselowej i znakowej (np. *InitGraph*, *TextMode*), przełączanie między trybem graficznym i tekstowym (np. *RestoreCrtMode*), ustanawianie okienek graficznych i tekstowych (np. *SetViewPort* i *Window*), a ponadto wyprowadzanie tekstów (np. *Write*), wykreślanie tekstów (np. *OutText*) i wykreślanie podstawowych obiektów graficznych (np. *Circle*).

Jeśli argumenty podprogramu nie naruszają wymagań składniowych, ale są dobrane niewłaściwie, to wykonanie podprogramu nie wywołuje żadnych

skutków. Jeśli argumenty są dobrane właściwie, ale wykonanie podprogramu okaże się niepomyślne, to wykonywanie programu jest kontynuowane. Zbadanie przyczyny niepomyślnego wykonania operacji graficznej pozostawia się programującemu. Zadanie to ułatwia funkcja *GraphResult*, której działanie (przedstawione szczegółowo dalej) może być porównane do działania funkcji *IOResult*.

Rezultatem funkcji *GraphResult* jest dana typu *integer*. Jeśli wykonanie podprogramu było pomyślne, to dana ta ma wartość 0. W przeciwnym razie ma wartość ujemną, identyfikującą przyczynę niepowodzenia. Ponieważ dwukrotne wywołanie funkcji *GraphResult* (bez wykonania między tymi wywołaniami operacji graficznej) powoduje, że jej drugim rezultatem jest zawsze dana o wartości 0, zaleca się więc przypisanie pierwszego rezultatu zmiennej pomocniczej (analogicznie do typowego użycia funkcji *IOResult*).

Tryb tekstowy

Wykonanie każdego programu rozpoczyna się w trybie tekstowym. Znaki wyprowadzane na ekran mogą być białe na czarnym tle, czarne na białym tle, migoczące i kolorowe. Wyprowadzanie znaków odbywa się za pośrednictwem plików skojarzonych z konsolą.

Okienko tekstowe

Po rozpoczęciu wykonywania programu domniemanym okienkiem tekstowym jest cały ekran. Lewy górny narożnik ekranu tekstowego ma współrzędne (1,1), a prawy dolny ma współrzędne (*col*,*lin*), gdzie *col* jest liczbą kolumn, a *lin* jest liczbą wierszy ekranu. Na przykład dla karty CGA w trybie C40 są to współrzędne (40,25). Z okienkiem jest związany widoczny kursor, którego położenie może być zmieniane za pomocą podprogramu *GotoXY*. Zmiana okienka tekstowego może być dokonana za pomocą podprogramu *Window*. Argumentami tego podprogramu są współrzędne ekranowe. Najmniejsze okienko ma wymiary 1 × 1 znak. Wszystkie operacje tekstowe dotyczą bieżącego okienka tekstowego, a współrzędne kursora tekstowego są zawsze liczone względem lewego górnego narożnika okienka. Okienko ma wszystkie właściwości ekranu monitora. W szczególności zachodzi w nim przewijanie pionowe.

Przykład. Przewijanie pionowe

```
program Scrolling;
uses
  Crt;
begin
  ClrScr;
  Window(1,1,5,3);
```

```
    Writeln('Jan');
    Writeln('Izabela')
  end.
```

Program jest ilustracją przewijania pionowego. Na ekranie pojawia się wiersz zawierający napis *Izabe* i wiersz zawierający się napis *la*. Na skutek przewinięcia, napis *Jan* znika z okienka tekstowego.

Kolory

Jeśli użyta karta oraz monitor zapewniają wyprowadzanie kolorowe, to mogą być niezależnie określone atrybuty koloru znaku i koloru tła (np. czerwony znak na zielonym tle). Ponadto można spowodować migotanie znaku oraz zwiększyć jego jasność. Atrybuty migotania, koloru tła, jasności i koloru znaku są dla każdego wyprowadzanego znaku określane na podstawie pól bitowych zmiennej *TextAttr*. Pola migotania i jasności są 1-bitowe, a pola koloru tła i koloru znaku są 3-bitowe. Przypisywanie danych wymienionym polom może odbywać się bezpośrednio, poprzez operacje na zmiennej *TextAttr*, albo pośrednio, za pomocą procedur *TextBackground* i *TextColor*. Pierwsza z tych procedur umożliwia określenie koloru tła, a druga umożliwia określenie koloru znaku, jego jasności i migotania. Argumenty wymienionych procedur są jak zwykle wyrażane za pomocą symboli, takich jak np. *Red* (czerwony), *Blink* (migotanie) itp.

Jeśli posłużono się kartą Hercules i monitorem monochromatycznym, to określanie atrybutów powinno odbywać się poprzez zmienną *TextAttr*. Przydatna może się wówczas okazać informacja, że najbardziej znaczący bit określa migotanie, trzy następne dotyczą tła, kolejny bit określa jasność, a trzy najmniej znaczące dotyczą znaku. Poza migotaniem i jasnością (wyrażonymi za pomocą bitów 1), mają sens jedynie następujące zestawy atrybutów

Pierwszy plan	*Tło*	*Skutek*
7	0	Białe znaki na czarnym tle
1	0	Białe podkreślone znaki na czarnym tle
0	7	Czarne znaki na białym tle
0	0	Czarne znaki na czarnym tle (oczywiście niewidoczne)

Przykład. Wyprowadzanie znaków w kolorach

```
program TextColors;
uses
  Crt;
begin
  ClrScr;
  TextColor(Red);
```

```
    TextBackground(Green);
    Write('Kajusia')
  end.
```

Na monitorze kolorowym sterowanym przez odpowiednią kartę graficzną (np. EGA albo CGA) pojawia się czerwony napis *Kajusia* na zielonym tle.

Tryb graficzny

Przełączanie systemu do trybu graficznego wymaga wykonania podprogramu *InitGraph* (jeśli posłużono się modułem *Graph*) albo wykonania podprogramu *GraphMode*, *GraphColorMode* albo *HiRes* (jeśli posłużono się modułem *Graph3*, a w systemie jest zainstalowana karta CGA albo karta stanowiąca jej rozszerzenie, np. EGA). Dalsze rozważania zostaną ograniczone do posłużenia się modułem *Graph*.

Przykład. Ustanowienie trybu graficznego

```
program Initialize;
uses
  Crt,Graph;
var
  Driver,Mode : integer;
begin
  Driver := Detect;
  InitGraph(Driver,Mode,'');
  PutPixel(0,0,1);
  repeat until KeyPressed;
  CloseGraph
end.
```

Wykonanie programu powoduje automatyczne rozpoznanie środowiska graficznego, przełączenie systemu do trybu graficznego i wykreślenie na ekranie jednego piksela.

Okienko graficzne

Po ustanowieniu trybu graficznego domniemanym okienkiem graficznym jest cały ekran. Lewy górny narożnik ekranu graficznego ma współrzędne (0,0), a prawy dolny ma współrzędne (*GetMaxX*, *GetMaxY*). Zmiana okienka graficznego może być dokonana za pomocą podprogramu *SetViewPort*. Jego argumentami są współrzędne okienka oraz wyrażenie logiczne, określające, czy wykresy wykraczające poza okienko mają być obcinane czy nie. Wszystkie operacje graficzne dotyczą bieżącego okienka graficznego, a współrzędne kursora graficznego są zawsze liczone względem lewego górnego narożnika

okienka. Kursor graficzny jest niewidoczny, ale jego bieżące współrzędne mogą być określone za pomocą procedur *GetX* i *GetY*. Współrzędne te są wartościami danych typu *integer* i mogą być zarówno dodatnie, jak i ujemne. Jeśli współrzędna *X* nie należy do przedziału 0..*GetX* lub współrzędna *Y* nie należy do przedziału 0..*GetY*, to kursor znajduje się poza okienkiem graficznym.

Przykład. Obcinanie na granicach okienka

```
program Clipping;
uses
  Crt,Graph;
var
  Driver,Mode : integer;
begin
  Driver:= HercMono;
  Mode:= HercMonoHi;
  InitGraph(Driver,Mode,'');
  SetViewPort(1,1,GetMaxX,GetMaxY,ClipOn);
  PutPixel(-1,-1,1);
  repeat until KeyPressed;
  CloseGraph
end.
```

Ponieważ procedurę *SetViewPort* wywołano z argumentem *ClipOn* (obcinaj), wykonanie procedury *PutPixel* nie wywołuje żadnych skutków. Gdyby argument *ClipOn* zastąpiono argumentem *ClipOff* (nie obcinaj), to w pobliżu lewego górnego narożnika okienka graficznego (ale poza okienkiem) zostałby wykreślony jeden piksel.

Wykreślanie tekstów

Teksty wyprowadzane na ekran graficzny są na nim wykreślane. Teksty mogą być dwóch rodzajów: bitowe i kreskowe. Każdy znak tekstu bitowego jest prostokątem o wymiarze podstawowym 8×8 bitów. Znak taki po powiększeniu nie wygląda zbyt dobrze. Znacznie lepszy efekt wizualny daje powiększony znak kreskowy. Jest on tworzony w sposób wektorowy i czytelność znaku nie zależy od jego rozmiaru. Przewidziano 4 kroje znaków kreskowych: indeksowy, potrójny, bezszeryfowy i gotycki. Znaki tych krojów mogą być nie tylko powiększane, lecz także rozciągane w poziomie i w pionie. Teksty złożone ze znaków dowolnego kroju mogą być wyprowadzane w poziomie (od lewej do prawej) albo w pionie (od dołu do góry). Alfabety krojów obejmują znaki podstawowego kodu ASCII, a alfabet kroju bitowego obejmuje znaki rozszerzonego kroju ASCII. Kroje kreskowe są przechowywane na dysku i ładowane dynamicznie do pamięci operacyjnej. W każdej chwili w pamięci może

znajdować się co najwyżej jeden automatycznie załadowany krój kreskowy. Zapewniono środki umożliwiające ładowanie krojów na żądanie (procedura *RegisterBGIFont*) oraz instalowanie ich w programie na stałe.

Przykład. Wykreślanie tekstu

```
program Texts;
uses
  Crt,Graph;
var
  Driver,Mode : integer;
begin
  Driver:= HercMono;
  Mode:= HercMonoHi;
  InitGraph(Driver,Mode,'');
  SetTextStyle(SmallFont,HorizDir,10);
  OutText('Jan ');
  SetTextStyle(GothicFont,HorizDir,8);
  OutText('Ewa ');
  SetTextStyle(TriplexFont,HorizDir,8);
  OutText('Iza ');
  Delay(2000);
  CloseGraph
end.
```

Wykonanie programu powoduje wykreślenie trzech tekstów: pierwszy jest wykreślany czcionką o kroju indeksowym (*SmallFont*), a następne odpowiednio gotykiem (*GothicFont*) i krojem potrójnym (*TriplexFont*). Znaki tekstu Jan są powiększone 10-krotnie, a pozostałe znaki 8-krotnie. Teksty są wyprowadzane w poziomie (*HorizDir*).

Wykreślanie obiektów graficznych

Biblioteka podprogramów graficznych zawiera podprogramy do wykreślania pikseli (np. *PutPixel*), odcinków (np. *Line*), łuków (np. *Arc*), okręgów (np. *Circle*), elips (np. *Ellipse*), a ponadto łamanych, wielokątów, słupków itp. Linie mogą być wykreślane jako grube albo cienkie, ciągłe albo przerywane, a obszary mogą być wypełniane wzorami standardowymi i projektowanymi.

Przykład. Wykreślenie wypełnionego okręgu

```
program Objects;
uses
  Crt,Graph;
var
  Driver,Mode : integer;
```

```
begin
  Driver:= HercMono;
  Mode:= HercMonoHi;
  InitGraph(Driver,Mode,'');
  Circle(GetMaxX div 2,GetMaxY div 2,300);
  FloodFill(GetMaxX shr 1,GetMaxY shr 1,GetMaxColor);
  repeat until KeyPressed;
  CloseGraph
end.
```

Wykonanie programu powoduje wykreślenie okręgu o promieniu 300 pikseli i środku w pobliżu środka ekranu, a następnie wypełnienie go wzorem domniemanym. Wykreślony obiekt jest obcięty na granicach okienka graficznego.

Kolory

Wykreślanie obiektów graficznych może być czarno-białe albo kolorowe. Wykreślanie czarno-białe dotyczy zazwyczaj kart graficznych nie uwzględniających koloru, jak np. karta Hercules. Oczywiście w takim przypadku kolor „biały" zależy od użytego luminoforu i może być w istocie np. zielony albo bursztynowy.

Jeśli jest możliwe wyświetlanie kolorowe (np. karta EGA albo CGA), to podczas instalowania sterownika graficznego można określić tryb jego pracy, a tym samym dostępną paletę kolorów.

Kolory palety (nie istnieje ona np. dla karty Hercules), są ponumerowane od 0 do GetMaxColor. Kolory te mogą być ustalone (np. karta CGA) albo dowolnie zmieniane (np. karta EGA). Jeśli kolory mogą być zmieniane, to przypisanie pozycji palety obranego koloru odbywa się za pomocą podprogramu *SetPalette*. Wybór numeru koloru do wykreślania obiektów zapewnia podprogram *SetColor*. Po wykonaniu podprogramu *SetColor*(*n*), wszystkie obiekty są wykreślane w kolorze związanym (sprzętowo albo programowo) z pozycją nr *n* palety kolorów. Programowe przypisanie tej pozycji innego koloru powoduje natychmiastową zmianę koloru rozpatrywanych obiektów.

Przykład. Wyświetlanie w kolorach

```
program GraphColors;
uses
  Crt,Graph;
var
  Driver,Mode : integer;
  Palette : PaletteType;
```

```
    i : shortint;
    Drop : char;
begin
    Driver := EGA;
    Mode := EgaLo;
    InitGraph(Driver,Mode,'');
    if GraphResult <> grOK then Halt(13);
    SetPalette(1,Red);
    SetColor(1);
    Line(0,0,GetMaxX,0);            {Red line}
    repeat until KeyPressed;
    Drop := ReadKey;
    SetPalette(1,Green);            {Green line}
    OutTextXY(0,0,'Press a key');   {Green text}
    repeat until KeyPressed;
    Drop := ReadKey;
    GetPalette(Palette);
    RestoreCrtMode;
    Writeln('Palette ');
    with Palette do
      for i := 0 to Size -1 do
        Writeln('No. ',i :2,' =>',Colors[i]];
end.
```

Program jest ilustracją zasady posługiwania się paletą kolorów karty EGA. Po ustanowieniu trybu graficznego i związaniu z pozycją nr 1 palety koloru czerwonego (*Red*) jest wykreślany poziomy odcinek linii prostej. Po naciśnięciu dowolnego klawisza klawiatury następuje zmiana koloru związanego z pozycją nr 1 palety na zielony (*Green*). Powoduje to, że zarówno już wykreślony odcinek, jak i tekst *Press a key* jest wyświetlany w tym kolorze. Na zakończenie wykonania programu następuje przełączenie systemu do trybu tekstowego i wyprowadzenie tablicy określającej identyfikatory kolorów przyporządkowane poszczególnym pozycjom palety. Bliższą analizę programu powinno ułatwić sięgnięcie do opisów podprogramów bibliotecznych.

Podprogramy graficzne

Arc (*Graph*)

Arc — wykreślenie łuku okręgu
(por. *Circle, Ellipse, GetArcCoords, GetAspectRatio, PieSlice*)

procedure *Arc*(*x*,*y* : *integer*;
StAngle,*EndAngle* : *word*;
Radius : *word*)

Wykreślenie łuku okręgu o środku w punkcie (*x*,*y*) i promieniu *Radius*, począwszy od kąta *StAngle*, a skończywszy na kącie *EndAngle*.

Uwagi. System musi być w trybie graficznym. Kąty *StAngle* i *EndAngle* są wyrażone w stopniach. Wykreślanie odbywa się w kierunku przeciwnym do ruchu wskazówek zegara. Dla *StAngle* = 0 i *EndAngle* = 360 jest wykreślany pełny okrąg.

Bar (*Graph*)

Bar — wykreślenie słupka
(por. *Bar3D*, *SetFillPattern*, *SetFillStyle*, *SetLineStyle*).

procedure *Bar*(*x1*,*y1* : *integer*;
x2,*y2* : *integer*)

Wykreślenie słupka (stanowiącego zapewne element wykresu słupkowego) jako wypełnionego prostokąta, którego przeciwległe wierzchołki mają współrzędne (*x1*,*y1*) i (*x2*,*y2*).

Uwagi. System musi być w trybie graficznym. Wzór i kolor wypełnienia prostokąta może być określony za pomocą procedury *SetFillStyle* albo *SetFillPattern*.

Bar3D (*Graph*)

Bar3D — wykreślenie słupka trójwymiarowego
(por. *Bar*, *GraphResult*, *SetFillPattern*, *SetFillStyle*, *SetLineStyle*)

procedure *Bar3D*(*x1*,*y1* : *integer*;
x2,*y2* : *integer*;
Depth : *word*;
TopFlag : *boolean*)

Wykreślenie trójwymiarowego słupka (stanowiącego zapewne element wykresu słupkowego) jako prostopadłościanu, którego przednia ściana jest wypełnionym prostokątem. Określenie przeciwległych wierzchołków prostokąta jako punktów o współrzędnych (*x1*,*y1*) i (*x2*,*y2*). Wyrażenie głębi prostopadłościanu za pomocą *Depth* i określenie za pomocą *TopFlag* czy ma być wykreślona jego górna powierzchnia.

Uwagi. System musi być w trybie graficznym. Wzór i kolor wypełnienia prostokąta może być określony za pomocą procedury *SetFillStyle* albo

SetFillPattern. Zaniechanie wykreślania górnej powierzchni umożliwia wykreślenie słupka składającego się z kilku klocków. Niepomyślne wykonanie procedury *Bar3D* powoduje, że rezultatem funkcji *GraphResult* jest dana o wartości

grNoScanMem — brak pamięci do wypełnienia obszaru metodą scan {−6}

Wartości trzeciego argumentu procedury mogą być wyrażone za pomocą następujących symboli

TopOff — bez wykreślania górnej powierzchni słupka {*false*}
TopOn — z wykreślaniem górnej powierzchni słupka {*true*}

Circle (*Graph*)

Circle — wykreślenie okręgu
(por. *Arc*, *Ellipse*, *GetArcCoords*, *GetAspectRatio*, *PieSlice*)

procedure *Circle*(*x*,*y* : *integer*;
Radius : *word*)

Wykreślenie okręgu o środku w punkcie (*x*,*y*) i promieniu *Radius*.

Uwagi. System musi być w trybie graficznym.

ClearDevice (*Graph*)

ClearViewPort — wyczyszczenie okienka graficznego
(por. *ClearViewPort*, *CloseGraph*, *InitGraph*, *RestoreCrtMode*, *SetGraphMode*)

procedure *ClearDevice*

Wyczyszczenie ekranu graficznego i usytuowanie kursora graficznego w lewym górnym narożniku bieżącego okienka graficznego.

Uwagi. System musi być w trybie graficznym.

ClearViewPort (*Graph*)

ClearViewPort — wyczyszczenie okienka graficznego
(por. *GetViewSettings*, *SetViewPort*)

procedure *ClearViewPort*

Wyczyszczenie bieżącego okienka graficznego, a następnie wypełnienie go kolorem przypisanym pierwszej pozycji palety.

Uwagi. System musi być w trybie graficznym. Niepomyślne wykonanie procedury *ClearViewPort* powoduje, że rezultatem funkcji *GraphResult* jest dana o wartości

grNoScanMem — brak pamięci do wypełnienia obszaru metodą scan {−6}.

CloseGraphMode (*Graph*)

CloseGraphMode — przywrócenie trybu tekstowego (por. *CloseGraph*, *DetectGraph*, *InitGraph*, *RegisterBGIDriver*, *RegisterBGIFont*, *RestoreCrtMode*, *SetGraphMode*)

procedure *CloseGraph*

Przywrócenie trybu tekstowego, zwolnienie pamięci przydzielonej automatycznie programowi sterownika graficznego oraz obranemu krojowi pisma.

Uwagi. System musi być w trybie graficznym. Pamięć przydzielona za pomocą procedur *RegisterBGIDriver* i *RegisterBGIFont* nie podlega zwolnieniu.

ClrEol (*Crt*)

ClrEol — wyczyszczenie końca wiersza (por. *ClrScr*, *Window*)

procedure *ClrEol*

Zastąpienie spacjami tych znaków wiersza, które występują między pozycją kursora a końcem wiersza.

Uwagi. System musi być w trybie znakowym. Jeśli kolorem tła nie jest czerń, to spacje są wyświetlane w kolorze określonym za pomocą procedury TextBackground. Zastępowanie znaków spacjami kończy się w chwili osiągnięcia prawego obrzeża bieżącego okienka tekstowego. Pozycja kursora tekstowego nie ulega zmianie.

ClrScr (*Crt*)

ClrScr — wyczyszczenie okienka tekstowego (por. *ClrEol*, *Window*)

procedure *ClrScr*

Wyczyszczenie bieżącego okienka tekstowego i usytuowanie kursora znakowego w lewym górnym narożniku tego okienka.

Uwagi. Jeśli kolorem tła nie jest czerń, to okienko zostanie wypełnione spacjami w kolorze określonym za pomocą procedury *TextBackground*.

DelLine (*Crt*)

DelLine — usunięcie wiersza
(por. *InsLine*, *Window*)

procedure *DelLine*

Usunięcie z okienka tekstowego wiersza wyróżnionego przez kursor. Przesunięcie wierszy położonych niżej o jeden wiersz do góry. Wstawienie na dole okienka jednego wiersza pustego.

Uwagi. Jeśli kolorem tła nie jest czerń, to wiersz pusty zostanie wypełniony spacjami w kolorze określonym przez procedurę *TextBackground*.

DetectGraph (*Graph*)

DetectGraph — rozpoznanie karty graficznej
(por. *GraphResult*, *InitGraph*)

procedure *DetectGraph* (**var** *GraphDriver* : *integer*;
var *GraphMode* : *integer*)

Rozpoznanie karty graficznej, a następnie przypisanie zmiennej *GraphDriver* danej o wartości równej numerowi karty, a zmiennej *GraphMode* danej o wartości równej numerowi domniemanego trybu graficznego.

Uwagi. Dane przypisane *GraphDriver* i *GraphMode* mogą być użyte w procedurze *InitGraph* do ustanowienia trybu graficznego. W istocie, wywołanie tej procedury z pierwszym argumentem *Detect* {0} powoduje niejawne wywołanie procedury *DetectGraph*. Numery kart i trybów mogą być wyrażone za pomocą następujących symboli

Symbol karty

Detect	— automatyczne rozpoznanie karty graficznej {0}
CGA	— karta CGA {1}
MCGA	— karta MCGA {2}
EGA	— karta EGA {3}
EGA64	— karta EGA64 {4}
EGAMono	— karta EGAMono {5}
IBM8514	— karta IBM 8514 {6}
HercMono	— karta Hercules {7}
ATT400	— karta ATT400 {8}
VGA	— karta VGA {9}
PC3270	— karta PC3270 {10}

Symbol trybu

CGAC0	— 320×200, paleta 0 {0}
CGAC1	— 320×200, paleta 1 {1}

CGAC2	–	320×200, paleta 2 {2}
CGAC3	–	320×200, paleta 3 {3}
CGAHi	–	640×200, 1 strona {4}
MCGAC0	–	320×200, paleta 0 {0}
MCGAC1	–	320×200, paleta 1 {1}
MCGAC2	–	320×200, paleta 2 {2}
MCGAC3	–	320×200, paleta 3 {3}
MCGAMed	–	640×200, 1 strona {4}
MCGAHi	–	640×480, 1 strona {5}
EGALo	–	640×200, 16 kolorów, 4 strony {0}
EGAHi	–	640×350, 16 kolorów, 2 strony {1}
EGA64Lo	–	640×200, 16 kolorów, 1 strona {0}
EGA64Hi	–	640×350, 4 kolory, 1 strona {1}
EGAMonoHi	–	640×350,
		64K na karcie, 1 strona {3}
		256K na karcie, 2 strony {3}
HercMonoHi	–	720×348, 2 strony {0}
ATT400C0	–	320×200, paleta 0 {0}
ATT400C1	–	320×200, paleta 1 {1}
ATT400C2	–	320×200, paleta 2 {2}
ATT400C3	–	320×200, paleta 3 {3}
ATT400Med	–	640×200, 1 strona {4}
ATT400Hi	–	640×400, 1 strona {5}
VGALo	–	640×200, 16 kolorów, 4 strony {0}
VGAMed	–	640×350, 16 kolorów, 2 strony {1}
VGAHi	–	640×480, 16 kolorów, 1 strona {2}
PC3270Hi	–	720×350, 1 strona {0}
IBM8514Lo	–	640×480, 256 kolorów
IBM8514Hi	–	1024×768, 256 kolorów

Niepomyślne wykonanie procedury *DetectGraph* powoduje, że rezultatem funkcji *GraphResult* jest dana o wartości

grNotDetected – brak karty graficznej {–2}

DrawPoly (*Graph*)

DrawPoly – wykreślenie linii łamanej
(por. *FillPoly*, *GetLineSettings*, *GraphResult*, *SetColor*, *SetLineStyle*)

procedure *DrawPoly*(*NumPoints* : *word*;
var *PolyPoints*)

Wykreślenie linii łamanej składającej się z *NumPoints* punktów, których

współrzędne (*x,y*) następują kolejno po sobie w obszarze pamięci zlokalizowanym przez *PolyPoints.*

Uwagi. System musi być w trybie graficznym. Każda para współrzędnych jest określona przez dwie dane typu *integer.* Wykreślenie wielokąta wymaga podania ostatniej pary współrzędnych identycznej z pierwszą.

Ellipse (*Graph*)

Ellipse — wykreślenie elipsy
(por. *Arc, Circle, GetArcCoords, GetAspectRatio, PieSlice*)

procedure *Ellipse*(*x,y* : *integer*;
StAngle,EndAngle : *word*;
xRadius,yRadius : *word*)

Wykreślenie łuku elipsy o środku w punkcie (*x,y*) i półosiach *xRadius* i *yRadius*, począwszy od kąta *StAngle*, a skończywszy na kącie *EndAngle.*

Uwagi. System musi być w trybie graficznym. Kąty *StAngle* i *EndAngle* są wyrażone w stopniach. Wykreślanie odbywa się w kierunku przeciwnym do ruchu wskazówek zegara. Dla *StAngle* = 0 i *EndAngle* = 360 jest wykreślana pełna elipsa.

FillPoly (*Graph*)

FillPoly — wypełnienie wielokąta
(por. *DrawPoly, GetFillSettings, GetLineSettings, GraphResult, SetFillPattern, SetFillStyle, SetLineStyle*)

procedure *FillPoly*(*NumPoints* : *word*;
var *PolyPoints*)

Wypełnienie zadanym wzorem, wielokąta o NumPoints wierzchołkach, których współrzędne znajdują się w obszarze pamięci zlokalizowanym przez *PolyPoints.*

Uwagi. System musi być w trybie graficznym. Każda para współrzędnych jest określona przez dwie dane typu *integer.* Wzór i kolor wypełnienia prostokąta może być określony za pomocą procedury *SetFillStyle* albo *SetFillPattern.* Niepomyślne wykonanie procedury *FillPoly* powoduje, że rezultatem funkcji *GraphResult* jest dana o wartości

grNoScanMem — brak pamięci do wypełnienia obszaru metodą scan {−6}

FloodFill (*Graph*)

FloodFill — wypełnienie obszaru zadanym wzorem (por. *FillPoly*, *SetFillPattern*, *SetFillStyle*)

procedure *FloodFill*(*x*,*y* : *integer*;
Border : *word*)

Wypełnienie zadanym wzorem wnętrza obszaru obejmującego punkt o współrzędnych (*x*,*y*), ograniczonego linią, której numer jest określony przez *Border*.

Uwagi. System musi być w trybie graficznym. Jeśli punkt o współrzędnych (*x*,*y*) znajduje się poza obszarem, to wypełnieniu podlega zewnętrze obszaru ograniczone obrzeżem okienka graficznego. Wykonanie procedury kończy się przedwcześnie po wprowadzeniu dwóch linii pustych. Wzór i kolor wypełnienia obszaru może być określony za pomocą procedury *SetFillStyle* albo *SetFillPattern*. Niepomyślne wykonanie procedury *FloodFill* powoduje, że rezultatem funkcji *GraphResult* jest dana o wartości

grNoFloodMem — brak pamięci do wypełnienia obszaru metodą flood {−7}

GetArcCoords (*Graph*)

GetArcCoords — określenie współrzędnych łuku (por. *Arc*, *Ellipse*, *PieSlice*)

procedure *GetArcCoords*(**var** *ArcCoords* : *ArcCoordsType*)

Przypisanie zmiennej *ArcCoords* danej rekordowej typu *ArcCoordsType* określającej parametry łuku okręgu wykreślonego za pomocą procedury *Arc*.

Uwagi. System musi być w trybie graficznym. Typ *ArcCoordsType* jest zdefiniowany następująco

```
type
  ArcCoordsType = record
                    x,y : integer;
                    xStart,yStart : integer;
                    xEnd,yEnd : integer
                  end;
```

Pola *x* i *y* określają współrzędne środka okręgu, pola *xStart* i *yStart* określają współrzędne początku łuku, a pola *xEnd* i *yEnd* — współrzędne końca łuku.

GetAspectRatio (*Graph*)

GetAspectRatio — wyznaczenie aspektu ekranu (por. *Arc*, *Circle*, *GetMaxX*, *GetMaxY*, *PieSlice*)

procedure *GetAspectRatio*(**var** *xAsp,yAsp* : *word*)

Przypisanie zmiennym *xAsp* i *yAsp* danych typu *word*, na podstawie których można określić aspekt ekranu.

Uwagi. System musi być w trybie graficznym. Posłużenie się aspektem wyrażonym jako *xAsp/yAsp* umożliwia wykreślanie prostokątów wyglądających jak kwadraty i elips wyglądających jak okręgi.

GetBkColor (*Graph*)

GetBkColor — określenie koloru tła
(por. *GetColor*, *GetPalette*, *InitGraph*,
SetAllPalette, *SetBkColor*,
[illegible] *SetPalette*)

function *GetBkColor* : *word*

Utworzenie danej typu *word*, o wartości równej numerowi tej pozycji palety, która określa kolor tła.

Uwagi. System musi być w trybie graficznym. Jeśli rezultatem funkcji jest dana o wartości *n*, to kolor tła jest określony przez *n*-tą pozycję palety. W przypadku karty Hercules kolor tła może być tylko czarny, a rezultatem funkcji *GetBkColor* jest dana o wartości 0.

GetColor (*Graph*)

GetColor — udostępnienie numeru koloru
(por. *GetBkColor*, *GetPalette*, *InitGraph*,
SetAllPalette, *SetColor*, *SetPalette*)

function *GetColor* : *word*

Utworzenie danej typu *word* o wartości równej numerowi koloru używanego do wykreślania linii.

Uwagi. System musi być w trybie graficznym. Jeśli rezultatem funkcji *GetColor* jest dana o wartości *n*, to bieżący kolor jest określony przez *n*-tą pozycję palety. W przypadku karty Hercules, rezultat o wartości 0 oznacza kolor czarny, a rezultat o wartości 1 oznacza kolor biały.

GetFillPattern (*Graph*)

GetFillPattern — ujawnienie wzoru wypełniania obszarów
(por. *SetFillPattern*, *SetFillStyle*)

procedure *GetFillPattern*(**var** *Pattern* : *FillPatternType*)

Przypisanie zmiennej *Pattern* danej tablicowej określającej zaprojektowany przez użytkownika wzór wypełniania obszarów.

Uwagi. System musi być w trybie graficznym. Typ *FillPatternType* jest zdefiniowany następująco

type
FillPatternType = **array** [1..8] **of** *byte*;

Jeśli przed wywołaniem procedury *GetFillPattern* nie wywołano procedury *SetFillPattern*, to poszczególnym elementom *Pattern* zostaną przypisane dane o wartości $ff.

GetFillSettings (*Graph*)

GetFillSettings – ujawnienie wzoru wypełniania obszarów (por. *FillPoly*, *SetFillPattern*, *SetFillStyle*)

procedure *GetFillSettings*(**var** *FillInfo* : *FillSettingsType*)

Przypisanie zmiennej *FillInfo* danej rekordowej typu *FillSettingsType*, określającej wzór do wypełniania obszarów oraz numer koloru do wykreślenia linii.

Uwagi. System musi być w trybie graficznym. Typ *FillSettingsType* jest zdefiniowany następująco

type
FillSettingsType = **record**
Pattern : *word*;
Color : *word*
end;

Pole *Pattern* podaje numer wzoru wybranego za pomocą procedury *SetFillStyle* (0..11) albo określonego za pomocą procedury *SetFillPattern* (12), a pole *Color* podaje numer koloru, który będzie użyty do wypełniania obszarów. Numery wzorów wypełniających mogą być wyrażone za pomocą następujących symboli:

EmptyFill – wypełnienie kolorem tła {0}
SolidFill – wypełnienie ciągłe {1}
LineFill – wypełnienie pogrubionymi liniami poziomymi {2}
LtSlashFill – wypełnienie liniami pochyłymi {3}
SlashFill – wypełnienie pogrubionymi liniami pochyłymi {4}
BkSlashFill – wypełnienie pogrubionymi liniami ukośnymi {5}
LtSkSlashFill – wypełnienie liniami ukośnymi {6}
HatchFill – wypełnienie siatką pionową {7}
xHatchFill – wypełnienie siatką ukośną {8}
InterleaveFill – wypełnienie liniami splecionymi {9}
WideDotFill – wypełnienie kropkami {10}
CloseDotFill – wypełnienie zagęszczonymi kropkami {11}

GetGraphMode (*Graph*)

GetGraphMode – ujawnienie bieżącego trybu graficznego (por. *ClearDevice*, *DetectGraph*, *InitGraph*, *RestoreCrtMode*, *SetGraphMode*)

function *GetGraphMode* : *integer*

Utworzenie danej typu *integer*, o wartości równej numerowi bieżącego trybu graficznego.

Uwagi. System musi być w trybie graficznym. Rezultatem jest dana o wartości z przedziału 0..5, określająca numer trybu ustanowionego za pomocą procedury *InitGraph* albo *SetGraphMode*. Numery mogą być wyrażone za pomocą następujących symboli:

CGAC0	–	320 × 200, paleta 0 {0}
CGAC1	–	320 × 200, paleta 1 {1}
CGAC2	–	320 × 200, paleta 2 {2}
CGAC3	–	320 × 200, paleta 3 {3}
CGAHi	–	640 × 200, 1 strona {4}
MCGAC0	–	320 × 200, paleta 0 {0}
MCGAC1	–	320 × 200, paleta 1 {1}
MCGAC2	–	320 × 200, paleta 2 {2}
MCGAC3	–	320 × 200, paleta 3 {3}
MCGAMed	–	640 × 200, 1 strona {4}
MCGAHi	–	640 × 480, 1 strona {5}
EGALo	–	640 × 200, 16 kolorów, 4 strony {0}
EGAHi	–	640 × 350, 16 kolorów, 2 strony {1}
EGA64Lo	–	640 × 200, 16 kolorów, 1 strona {0}
EGA64Hi	–	640 × 350, 4 kolory, 1 strona {1}
EGAMonoHi	–	640 × 350, 64K na karcie, 1 strona {3} 256K na karcie, 2 strony {3}
HercMonoHi	–	720 × 348, 2 strony {0}
ATT400C0	–	320 × 200, paleta 0 {0}
ATT400C1	–	320 × 200, paleta 1 {1}
ATT400C2	–	320 × 200, paleta 2 {2}
ATT400C3	–	320 × 200, paleta 3 {3}
ATT400Med	–	640 × 200, 1 strona {4}
ATT400Hi	–	640 × 400, 1 strona {5}
VGALo	–	640 × 200, 16 kolorów, 4 strony {0}
VGAMed	–	640 × 350, 16 kolorów, 2 strony {1}
VGAHi	–	640 × 480, 16 kolorów, 1 strona {2}

PC3270Hi – 720×350, 1 strona
IBM8514Lo – 640×480, 256 kolorów
IBM8514Hi – 1024×768, 256 kolorów

GetImage (*Graph*)

GetImage – zapamiętanie obrazu
(por. *ImageSize*, *PutImage*)

procedure *GetImage* (*x1,y1* : *integer*;
x2,y2 : *integer*;
var *BitMap*)

Wyodrębnienie – w obrębie bieżącego okienka graficznego – prostokątnego obszaru, którego przeciwległe wierzchołki mają współrzędne (*x1,y1*) i (*x2,y2*), a następnie zapamiętanie obrazu zawartego w tym prostokącie w obszarze zlokalizowanym przez *BitMap.*

Uwagi. System musi być w trybie graficznym. Zaleca się, aby rozmiar obszaru był określony za pomocą procedury *ImageSize.*

GetLineSettings (*Graph*)

GetLineSettings – ujawnienie parametrów linii
(por. *SetLineStyle*)

procedure *GetLineSettings* (**var** *LineInfo* : *LineSettingsType*)

Przypisanie zmiennej *LineInfo* danej rekordowej typu *LineSettingsType*, określającej rodzaj, wzór i grubość linii.

Uwagi. System musi być w trybie graficznym. Typ *LineSettingsType* jest zdefiniowany następująco

```
type
  LineSettingsType = record
                       LineStyle : word;
                       Pattern : word;
                       Thickness : word
                     end;
```

Pole *LineStyle* określa numer rodzaju linii. Jeśli numerem tym jest *UserBitLn*, to pole *Pattern* określa wzór linii. Pole *Thickeness* określa grubość linii. Wartości danych przypisanych polu *LineStyle* mogą być wyrażone za pomocą następujących symboli

SolidLn – linia ciągła {0}
DottedLn – linia kropkowana {1}
CenterLn – linia centrowana {2}

DashedLn — linia przerywana {3}
UserBitLn — linia zdefiniowana {4}

Wartości danych przypisanych polu *Thickness* mogą być wyrażone za pomocą następujących symboli

NormWidth — linia cienka {1}
ThickWidth — linia pogrubiona {3}

Polu *Pattern* może być przypisany dowolny wzór 16-bitowy.

GetMaxColor (*Graph*)

GetMaxColor — ujawnienie maksymalnego numeru koloru (por. *SetColor*)

function *GetMaxColor : word*

Utworzenie danej typu *word*, o wartości równej maksymalnemu numerowi koloru.

Uwagi. System musi być w trybie graficznym.

GetMaxX (*Graph*)

GetMaxX — ujawnienie maksymalnej współrzędnej poziomej ekranu graficznego
(por. *GetMaxY, GetX, GetY, MoveTo*)

function *GetMaxX : integer*

Utworzenie danej typu *integer* o wartości równej maksymalnej współrzędnej poziomej ekranu graficznego.

Uwagi. System musi być w trybie graficznym. Współrzędne poziome należą do przedziału 0..*GetMaxX*.

GetMaxY (*Graph*)

GetMaxY — ujawnienie maksymalnej współrzędnej pionowej ekranu graficznego
(por. *GetMaxX, GetX, GetY, MoveTo*)

function *GetMaxY : integer*

Utworzenie danej typu *integer* o wartości równej maksymalnej współrzędnej pionowej ekranu graficznego.

Uwagi. System musi być w trybie graficznym. Współrzędne pionowe należą do przedziału 0..*GetMaxY*.

GetModeRange (*Graph*)

GetModeRange – ujawnienie zakresu trybów graficznych (por. *DetectGraph*, *InitGraph*)

procedure *GetModeRange*(*GraphDriver* : *integer*;
var *LoMode* : *integer*;
var *HiMode* : *integer*)

Przypisanie *LoMode* najniższego, a *HiMode* najwyższego numeru trybu karty graficznej o numerze *GraphDriver*.

Uwagi. System może być w trybie graficznym. Jeśli numer karty nie należy do przedziału 1..5 lub 7..10, to zmiennym *LoMode* i *HiMode* zostaną przypisane dane o wartości −1. Numery kart graficznych mogą być wyrażone za pomocą następujących symboli:

CGA – karta CGA {1}
MCGA – karta MCGA {2}
EGA – karta EGA {3}
EGAMono – karta EGAMono {5}
IBM8514 – karta IBM8514 {6}
HercMono – karta Hercules {7}
ATT400 – karta ATT400 {8}
VGA – karta VGA {9}
PC3270 – karta PC3270 {10}

GetPalette (*Graph*)

GetPalette – ujawnienie bieżącej palety kolorów (por. *SetAllPalette*, *SetPalette*)

procedure *GetPalette*(**var** *Palette* : *PaletteType*)

Przypisanie zmiennej *Palette* danej rekordowej typu *PaletteType*, określającej rozmiar palety kolorów oraz identyfikatory kolorów przypisane poszczególnym pozycjom palety.

Uwagi. System musi być w trybie graficznym. Typ *PaletteType* jest zdefiniowany następująco:

```
type
  PaletteType = record
                  Size : byte;
                  Colors : array[0..15] of shortint
                end;
```

Pole *Size* określa rozmiar palety, a pole *Colors* określa identyfikatory kolorów przypisane kolejnym pozycjom palety. Identyfikatory kolorów mogą być wyrażone za pomocą następujących symboli

Black – czarny {0}
Blue – niebieski {1}
Green – zielony {2}
Cyan – turkusowy {3}
Red – czerwony {4}
Magenta – karmazynowy {5}
Brown – brązowy {6}
LightGray – jasnoszary {7}
DarkGray – ciemnoszary {8}
LightBlue – jasnoniebieski {9}
LightGreen – jasnozielony {10}
LightCyan – jasnoturkusowy {11}
LightRed – jasnoczerwony {12}
LightMagenta – jasnokarmazynowy {13}
Yellow – żółty {14}
White – biały {15}

GetPixel (*Graph*)

GetPixel – ujawnienie numeru koloru piksela (por. *GetImage*, *PutImage*, *PutPixel*)

function *GetPixel*(*x*,*y* : *integer*) : *word*

Utworzenie danej typu *word*, o wartości równej numerowi koloru piksela wyświetlanego w punkcie (*x*,*y*).

Uwagi. System musi być w trybie graficznym.

GetTextSettings (*Graph*)

GetTextSettings – ujawnienie parametrów wykreślania tekstów (por. *InitGraph*, *SetTextJustify*, *SetTextStyle*, *TextHeight*, *TextWidth*)

procedure *GetTextSettings*(**var** *TextInfo* : *TextSettingsType*)

Przypisanie zmiennej *TextInfo* danej rekordowej typu *TextSettingsType*, określającej krój czcionki, kierunek wykreślania tekstu i sposób jego wyrównania.

Uwagi. System musi być w trybie graficznym. Typ *TextSettingsType* jest zdefiniowany następująco

```
type
  TextSettingsType = record
                       Font : word;
                       Direction : word;
                       CharSize : 1..10;
                       Horiz : word;
                       Vert : word
                     end;
```

Pole *Font* określa numer kroju czcionki; pole *Direction* określa kierunek wykreślania tekstów (*HorizDir* – poziomo, *VertDir* – pionowo); pole *CharSize* określa mnożnik rozmiaru czcionki (jeśli *CharSize* = *UserCharSize*, to pionowy i poziomy mnożnik rozmiaru wynika z użycia procedury *SetUserCharSize*); pole *Horiz* określa sposób wyrównania tekstu w poziomie (*LeftText* – do lewej, *CenterText* – centrycznie, *RightText* – do prawej); pole *Vert* określa sposób wyrównania tekstu w pionie (*BottomText* – do dołu, *CenterText* – centrycznie, *TopText* – do góry). Wartości danych przypisanych polom *Font*, *Direction*, *Horiz* i *Vert* mogą być wyrażone za pomocą następujących symboli:

Pole Font

DefaultFont – krój domniemany {0}
TriplexFont – krój potrójny {1}
SmallFont – krój indeksowy {2}
SansSerifFont – krój bezszeryfowy {3}
GothicFont – krój gotycki {4}

Pole Direction

HorizDir – kierunek poziomy {0}
VertDir – kierunek pionowy {1}

Pole Horiz

LeftText – wyrównanie do lewej {0}
CenterText – wyrównanie centryczne {1}
RightText – wyrównanie do prawej {2}

Pole Vert

BottomText – wyrównanie do dołu {0}
CenterText – wyrówanie centryczne {1}
TopText – wyrównanie do góry {2}

GetViewSettings (*Graph*)

GetViewSettings – ujawnienie parametrów okienka graficznego (por. *SetViewPort*)

procedure *GetViewSettings* (**var** *ViewPor* : *ViewPortType*)

Przypisanie zmiennej *ViewPort* danej rekordowej typu *ViewPortType*, określającej współrzędne przeciwległych wierzchołków okienka graficznego oraz sposób traktowania wykresów wykraczających poza okienko.

Uwagi. System musi być w trybie graficznym. Typ *ViewPortType* jest zdefiniowany następująco

```
type
  ViewPortType = record
                   x1,y1 : integer;
                   x2,y2 : integer;
                   Clip : boolean
                 end;
```

Pola *x1*, *y1* i *x2*, *y2* określają współrzędne przeciwległych wierzchołków okienka graficznego, a pole *Clip* określa, czy wykresy mają być obcinane na obrzeżach okienka. Wartości danych przypisanych polu *Clip* mogą być wyrażone za pomocą następujących symboli.

ClipOff — bez obcinania {*false*}
ClipOn — z obcinaniem {*true*}

GetX (*Graph*)

GetX — ujawnienie poziomej współrzędnej kursora graficznego (por. *GetViewSettings*, *GetY*, *InitGraph*, *MoveTo*, *SetViewPort*)

function *GetX* : *integer*

Utworzenie danej typu *integer* o wartości równej poziomej współrzędnej kursora graficznego.

Uwagi. System musi być w trybie graficznym. Współrzędna jest określona względem bieżącego okienka graficznego.

GetY (*Graph*)

GetY — ujawnienie pionowej współrzędnej kursora graficznego (por. *GetViewSettings*, *GetX*, *InitGraph*, *MoveTo*, *SetViewPort*)

function *GetY* : *integer*

Utworzenie danej typu *integer* o wartości równej poziomej współrzędnej kursora graficznego.

Uwagi. System musi być w trybie graficznym. Współrzędna jest określona względem bieżącego okienka graficznego.

GotoXY (*Crt*)

GotoXY – przemieszczenie kursora tekstowego (por. *WhereX*, *WhereY*, *Window*)

procedure *GotoXY*(*x*,*y* : *byte*)

Przemieszczenie kursora tekstowego na pozycję o współrzędnych (*x*,*y*).

Uwagi. Współrzędne są określane względem bieżącego okienka tekstowego. Lewy górny narożnik tego okienka ma współrzędne (1,1). Jeśli nowe współrzędne miałyby wykraczać poza okienko, to wykonanie procedury nie wywoła żadnych skutków.

GraphDefaults (*Graph*)

GraphDefaults – przywrócenie domniemanych parametrów graficznych (por. *InitGraph*)

procedure *GraphDefaults*

Ustanowienie okienka graficznego obejmującego cały ekran, przemieszczenie kursora graficznego do jego lewego górnego narożnika oraz ustanowienie wszystkich domniemanych parametrów graficznych (palety, kroju czcionek, postaci linii, sposobu wypełniania obszarów i wyrównywania tekstów, rozmiaru czcionek, koloru tła i koloru pierwszego planu).

Uwagi. System musi być w trybie graficznym.

GraphErrorMsg (*Graph*)

GraphErrorMsg – ujawnienie komunikatu o przebiegu wykonania podprogramu graficznego (por. *GraphResult*, *DetectGraph*, *InitGraph*)

function *GraphErrorMsg* (*ErrorCode* : *integer*) : *string*

Utworzenie danej typu *string*, złożonej ze znaków tworzących komunikat o numerze *ErrorCode*.

Uwagi. System musi być w trybie graficznym. Numery komunikatów mogą być wyrażone za pomocą następujących symboli:

grOK	– pomyślne wykonanie operacji graficznej {0}
grNoInitGraph	– nie ustanowiono trybu graficznego {1}
grNotDetected	– nie wykryto karty graficznej {–2}
grFileNotFound	– brak zbioru zawierającego sterownik graficzny {–3}
grInvalidDriver	– niewłaściwy sterownik graficzny {–4}
grNoLoadMem	– brak pamięci do załadowania sterownika {–5}

grNoScanMem – brak pamięci do wypełnienia obszaru metodą *scan* {–6}

grNoFloodMem – brak pamięci do wypełnienia obszaru metoda *flood* {–7}

grFontFound – brak zbioru zawierającego definicję kroju czcionek {–8}

grNoFontMem – brak pamięci do załadowania kroju kreskowego {–9}

grInvalidMode – niewłaściwie dobrany tryb graficzny {–10}

grError – inne błędy (np. nadmierna liczba zgłoszonych krojów kreskowych) {–11}

grIOError – błąd operacji wejścia/wyjścia {–12}

grInvalidFont – niewłaściwy krój czcionek {–13}

grInvalidFontNum – niewłaściwy numer kroju czcionek {–14}

grInvalidDeviceNum – niewłaściwy numer urządzenia {–15}

GraphResult (*Graph*)

GraphResult – ujawnienie kodu powrotu ostatniej operacji graficznej (por. *GraphErrorMsg*)

function *GraphResult* : *integer*

Utworzenie danej typu *integer* o wartości określającej kod powrotu ostatniej operacji graficznej.

Uwagi. System musi być w trybie graficznym. Rezultat o wartości 0 oznacza pomyślne, a rezultat o wartości ujemnej niepomyślne wykonanie operacji. Kody powrotu mogą być wyrażone za pomocą następujących symboli

grOK – pomyślne wykonanie operacji graficznej {0}

grNoInitGraph – nie ustanowiono trybu graficznego {1}

grNotDetected – nie wykryto karty graficznej {–2}

grFileNotFound – brak zbioru zawierającego sterownik graficzny {–3}

grInvalidDriver – niewłaściwy sterownik graficzny {–4}

grNoLoadMem – brak pamięci do załadowania sterownika {–5}

grNoScanMem – brak pamięci do wypełnienia obszaru metodą scan {–6}

grNoFloodMem – brak pamięci do wypełnienia obszaru metodą flood {–7}

grFontFound – brak zbioru zawierającego definicję kroju czcionek {–8}

grNoFontMem – brak pamięci do załadowania kroju kreskowego {–9}

grInvalidMode	–	niewłaściwie dobrany tryb graficzny {–10}
grError	–	inne błędy (np. nadmierna liczba zgłoszonych krojów kreskowych) {–11}
grIOError	–	błąd operacji wejścia/wyjścia {–12}
grInvalidFont	–	niewłaściwy krój czcionek {–13}
grInvalidFontNum	–	niewłaściwy numer kroju czcionek {–14}
grInvalidDeviceNum	–	niewłaściwy numer urządzenia {–15}

Ponieważ powtórne wywołanie funkcji *GraphResult* (bez wykonania w międzyczasie operacji graficznej) powoduje udostępnienie danej o wartości 0, zaleca się przypisać rezultat funkcji zmiennej pomocniczej.

HighVideo (*Crt*)

HighVideo – zwiększenie jasności znaków (por. *LowVideo*, *NormVideo*, *TextBackground*, *TextColor*)

procedure *HighVideo*

Ustawienie bitu jasności zmiennej *TextAttr* i tym samym spowodowanie, że następne znaki wyprowadzane na ekran będą rozjaśnione.

Uwagi. Ustawienie bitu jasności powoduje odwzorowanie ciemnych kolorów pierwszego planu na jasne:

Kolor ciemny	*Kolor jasny*	
Black	*DarkGray*	(czarny/ciemnoszary)
Blue	*LightBlue*	(niebieskie)
Green	*LightGreen*	(zielone)
Cyan	*LightCyan*	(turkusowe)
Red	*LightRed*	(czerwone)
Magenta	*LightMagenta*	(karmazynowe)
Brown	*LightBrown*	(brązowe)
LightGray	*White*	(jasnoszary/biały)

ImageSize (*Graph*)

ImageSize – wyznaczenie rozmiaru pamięci do zapamiętania obrazu graficznego (por. *GetImage*, *PutImage*)

function *ImageSize* (*x1*,*y1* : *integer*;
x2,*y2* : *integer*) : *word*

Utworzenie danej typu *word*, o wartości równej liczbie bajtów pamięci niezbędnych do zapamiętania obrazu graficznego wyświetlanego w prostokącie, którego przeciwległe wierzchołki mają współrzędne (*x1*,*y1*) i (*x2*,*y2*).

Uwagi. System musi być w trybie graficznym.

InitGraph (*Graph*)

InitGraph – ustanowienie trybu graficznego (por. *CloseGraph*, *DetectGraph*, *GraphResult*, *RestoreCrtMode*, *SetGraphMode*)

procedure *InitGraph*(**var** *GraphDriver* : *integer*;
var *GraphMode* : *integer*;
PathToDriver : *string*)

Odszukanie w katalogu *PathToDriver* zbioru zawierającego program sterownika graficznego karty o numerze *GraphDriver*, sprowadzenie programu do pamięci operacyjnej i ustanowienie trybu graficznego *GraphMode*.

Uwagi. Jeśli nazwa katalogu jest pusta, to zbiór jest poszukiwany w katalogu bieżącym. Jeśli *GraphDriver* = 0, to następuje automatyczne rozpoznanie środowiska graficznego, a zmiennym *GraphDriver* i *GraphMode* są przypisywane dane o wartościach określających numer sterownika i numer trybu. Jeśli *GraphDriver*<>0, to *GraphDriver* i *GraphMode* muszą poprawnie określać numer karty i numer trybu. Pamięć operacyjna wymagana przez sterownik jest przydzielana na stercie i zwalana w ramach wykonania procedury *CloseGraph*. Numery kart i trybów mogą być wyrażone za pomocą następujących symboli.

Symbol karty

Detect – automatyczne rozpoznanie karty graficznej {0}
CGA – karta CGA {1}
MCGA – karta MCGA {2}
EGA – karta EGA {3}
EGA6 – karta EGA64 {4}
EGAMono – karta EGAMono {5}
IBM8514 – karta IBM 8514 {6}
HercMono – karta Hercules {7}
ATT400 – karta ATT400 {8}
VGA – karta VGA {9}
PC3270 – karta PC3270 {10}

Symbol trybu

CGAC0 – 320×200, paleta 0 {0}
CGAC1 – 320×200, paleta 1 {1}
CGAC2 – 320×200, paleta 2 {2}
CGAC3 – 320×200, paleta 3 {3}
CGAHi – 640×200, 1 strona {4}
MCGAC0 – 320×200, paleta 0 {0}

MCGAC1 – 320×200, paleta 1 {1}
MCGAC2 – 320×200, paleta 2 {2}
MCGAC3 – 320×200, paleta 3 {3}
MCGAMed – 640×200, 1 strona {4}
MCGAHi – 640×480, 1 strona {5}
EGALo – 640×200, 16 kolorów, 4 strony {0}
EGAHi – 640×350, 16 kolorów, 2 strony {1}
EGA64Lo – 640×200, 16 kolorów, 1 strona {0}
EGAMonoHi – 640×350,
64K na karcie, 1 strona {3}
256K na karcie, 2 strony {3}
HercMonoHi – 720×348, 2 strony {0}
ATT400C0 – 320×200, paleta 0 {0}
ATT400C1 – 320×200, paleta 1 {1}
ATT400C2 – 320×200, paleta 2 {2}
ATT400C3 – 320×200, paleta 3 {3}
ATT400Med – 640×200, 1 strona {4}
ATT400Hi – 640×400, 1 strona {5}
VGALo – 640×200, 16 kolorów, 4 strony {0}
VGAMed – 640×350, 16 kolorów, 2 strony {1}
VGAHi – 640×480, 16 kolorów, 1 strona {2}
PC3270Hi – 720×350, 1 strona {0}
IBM8514Lo – 640×480, 256 kolorów
IBM8514Hi – 1024×768, 256 kolorów

Niepomyślne wykonanie procedury *InitGraph* powoduje, że rezultatem funkcji *GraphResult* jest dana o jednej z następujących wartości

grNotDetected – nie wykryto karty graficznej {–2}
grFileNotFound – brak zbioru zawierającego sterownik graficzny {–3}
grInvalidDriver – niewłaściwy sterownik graficzny {–4}
grNoLoadMem – brak pamięci do załadowania sterownika {–5}
grInvalidMode – niewłaściwie dobrany tryb graficzny {–10}
grInvalidDeviceNum – niewłaściwy numer urządzenia {–15}

InsLine (*Ctr*)

InsLine – wstawienie wiersza
(por. *DelLine, Window*)

procedure *InsLine*

Wstawienie w okienku tekstowym, bezpośrednio pod wierszem wyróżnionym

przez kursor, jednego wiersza pustego. Przesunięcie wierszy położonych poniżej wyróżnionego o jeden do dołu. Usunięcie z okienka, wiersza położonego najniżej.

Uwagi. Jeśli kolorem tła nie jest czerń, to wiersz pusty zostanie wypełniony spacjami w kolorze określonym za pomocą procedury *TextBackground.*

Line (*Graph*)

Line — wykreślenie odcinka linii prostej
(por. *LineTo, MoveTo, Rectangle, SetColor, SetLineStyle*)

procedure *Line(x1,y1 : integer;*
x2,y2) : integer)

Wykreślenie odcinka linii prostej łączącego punkt o współrzędnych (*x1,y1*) z punktem o współrzędnych (*x2,y2*).

Uwagi. System musi być w trybie graficznym. Współrzędne kursora graficznego nie ulegają zmianie.

LineRel (*Graph*)

LineRel — wykreślenie odcinka linii prostej
(por. *Line, LineTo, MoveRel, MoveTo, SetColor, SetLineStyle*)

procedure *LineRel(dx,dy : integer)*

Wykreślenie odcinka linii prostej, łączącego punkt wyróżniony przez kursor graficzny z punktem odległym od niego o *dx* w poziomie i *dx* w pionie. Zmiana wpółrzędnych kursora graficznego o *dx* w poziomie i *dy* w pionie.

Uwagi. System musi być w trybie graficznym.

LineTo (*Graph*)

LineTo — wykreślenie odcinka linii prostej
(por. *Line, LineRel, MoveTo, MoveRel, SetColor, SetLineStyle*)

procedure *LineTo(x,y : integer)*

Wykreślenie odcinka linii prostej, łączącego punkt wyróżniony przez kursor graficzny z punktem o współrzędnych (*x,y*). Przesunięcie kursora graficznego do punktu o współrzędnych (*x,y*).

Uwagi. System musi być w trybie graficznym.

LowVideo (*Crt*)

LowVideo – zmniejszenie jasności znaków
(por. *HighVideo*, *NormVideo*, *TextBackground*, *TextColor*)

procedure *LowVideo*

Wyzerowanie bitu jasności zmiennej *TextAttr* i tym samym spowodowanie, że następne znaki wyprowadzane na ekran będą przyciemnione.

Uwagi. Wyzerowanie bitu jasności powoduje odwzorowanie jasnych kolorów pierwszego planu na ciemne

Kolor jasny	*Kolor ciemny*	
DarkGray	*Black*	(ciemnoszary/czarny)
LightBlue	*Blue*	(niebieskie)
LightGreen	*Green*	(zielone)
LightCyan	*Cyan*	(turkusowe)
LightRed	*Red*	(czerwone)
LightMagenta	*Magenta*	(karmazynowe)
Yellow	*Brown*	(brązowe)
White	*LightGray*	(biały/jasnoszary)

MoveRel (*Graph*)

MoveRel – przemieszczenie kursora graficznego
(por. *LineRel*, *LineTo*, *MoveTo*)

procedure *MoveRel*(*dx*,*dy* : *integer*)

Zmiana współrzędnej poziomej kursora graficznego o *dx* i współrzędnej pionowej o *dy*.

Uwagi. System musi być w trybie graficznym.

MoveTo (*Graph*)

MoveTo – przemieszczenie kursora graficznego
(por. *LineRel*, *LineTo*, *MoveRel*)

procedure *MoveTo*(*x*,*y* : *integer*)

Przemieszczenie kursora graficznego do punktu o współrzędnych (*x*,*y*).

Uwagi. System musi być w trybie graficznym.

NormVideo (*Crt*)

NormVideo – odtworzenie atrybutu wyświetlania znaków
(por. *HighVideo*, *LowVideo*, *TextBackground*, *TextColor*)

procedure *NormVideo*

Przypisanie zmiennej *TextAttr* takiego atrybutu określającego sposób wyświetlania znaków, jaki obowiązywał w chwili rozpoczęcia wykonywania programu.

OutText (*Graph*)

OutText — wykreślenie tekstu
(por. *GetTextSettings*, *OutTextXY*, *SetTextJustify*, *SetTextStyle*, *TextHeight*, *TextWidth*)

procedure *OutText*(*TextString* : *string*)

Wykreślenie w pobliżu bieżącej pozycji kursora graficznego, tekstu określonego przez *TextString*.

Uwagi. System musi być w trybie graficznym. Tekst jest wykreślany czcionką określoną za pomocą procedury *SetTextStyle* i w kolorze określonym za pomocą procedury *SetColor*. Usytuowanie tekstu względem pozycji kursora graficznego może być określone za pomocą procedury *SetTextJustify*. Jeśli pierwszy argument tej procedury ma wartość *LeftText*, a tekst jest wyświetlany w poziomie, to pozioma współrzędna kursora graficznego jest przemieszczana o szerokość wyświetlanego tekstu. W pozostałych przypadkach pozycja kursora nie ulega zmianie. Jeśli tekst nie mieści się w bieżącym okienku graficznym, to jest obcinany na jego obrzeżach.

OutTextXY (*Graph*)

OutTextXY — wykreślenie tekstu
(por. *GetTextSettings*, *OutText*, *SetTextJustify*, *SetTextStyle*, *TextHeight*, *TextWidth*)

procedure *OutTextXY* (*x*,*y* : *integer*;
TextString : *string*)

Wykreślenie w pobliżu punktu o współrzędnych (*x*,*y*), tekstu określonego przez *TextString*.

Uwagi. System musi być w trybie graficznym. Pozycja kursora graficznego nie ulega zmianie. Tekst jest wykreślany czcionką określoną za pomocą procedury *SetTextStyle* i w kolorze określonym za pomocą procedury *SetColor*. Sposób usytuowania tekstu względem punktu o współrzędnych (*x*,*y*) może być określony za pomocą procedury *SetTextJustify*. Jeśli tekst nie mieści się w bieżącym okienku graficznym, to jest obcinany na jego obrzeżach.

PieSlice (*Graph*)

PieSlice — wykreślenie wycinka koła
(por. *Arc*, *Circle*, *Ellipse*, *GetArcCoords*, *GetAspectRatio*, *SetColor*, *SetFillPattern*, *SetFillStyle*)

procedure *PieSlice*(*x*,*y* : *integer*;
StAngle,*EndAngle* : *word*;
Radius : *word*)

Wykreślenie wycinka koła o środku w punkcie (*x*,*y*) i promieniu *Radius*, począwszy od kąta *StAngle*, a skończywszy na kącie *EndAngle*. Wypełnienie wycinka wzorem określonym za pomocą procedury *SetFillStyle* albo *SetFillPattern*.

Uwagi. System musi być w trybie graficznym. Obranie kątów *StAngle* = 0 i *EndAngle* = 360 powoduje wykreślenie pełnego koła. Niepomyślne wykonanie procedury *PieSlice* powoduje, że rezultatem funkcji *GraphResult* jest dana o wartości

grNoScanMem — brak pamięci do wypełnienia obszaru metodą *scan* {−6}

PutImage (*Graph*)

PutImage — umieszczenie obrazu na ekranie (por. *GetImage*, *ImageSize*)

procedure *PutImage*(*x*,*y* : *integer*;
var BitMap;
Op : *word*)

Umieszczenie na ekranie prostokątnego obrazu graficznego zapamiętanego uprzednio w obszarze zlokalizowanym przez *BitMap*. Wykonanie tego w taki sposób, aby lewy górny narożnik prostokąta znalazł się w punkcie o współrzędnych (*x*,*y*), a nałożenie obrazu na ekran odbyło się zgodnie z wartością *Op*.

Uwagi. System musi być w trybie graficznym. Wykonanie procedury *PutImage* nie powoduje obcięcia obrazu na obrzeżach okienka graficznego. Jeśli obcięcie byłoby wymagane, to wykonanie procedury nie wywoła żadnych skutków. Wyjątek: jeśli dolnym obrzeżem okienka jest dolne obrzeże ekranu, to obraz wykraczający poza to obrzeże (i tylko poza nie) jest umieszczany na ekranie i obcinany. Sposób nałożenia obrazu na ekran (parametr *Op*) może być określony za pomocą następujących symboli

NormalPut — operacja mov {0}
xorPut — operacja xor {1}
orPut — operacja or {2}
andPut — operacja and {3}
notPut — operacja not {4}

PutPixel (*Graph*)

PutPixel — wyświetlenie piksela (por. *GetImage*, *GetPixel*, *PutImage*)

procedure *PutPixel*(*x*,*y* : *integer*;
Color : *word*)

Wyświetlenie w punkcie o współrzędnych (*x*,*y*) piksela w kolorze o numerze *Color*.

Uwagi. System musi być w trybie graficznym.

Rectangle (*Graph*)

Rectangle — wykreślenie prostokąta
(por. *Bar*, *Bar3D*, *SetColor*, *SetLineStyle*)

procedure *Rectangle*(*x1*,*y1* : *integer*;
x2,*y2* : *integer*)

Wykreślenie prostokąta, którego przeciwległe wierzchołki mają współrzędne (*x1*,*y1*) i (*x2*,*y2*).

Uwagi. System musi być w trybie graficznym.

RegisterBGIDriver (*Graph*)

RegisterBGDriver — zgłoszenie sterownika graficznego
(por. *RegisterBGIFont*)

function *RegisterBGIDriver*(*Driver* : *pointer*): *integer*

Zgłoszenie, że w obszarze pamięci operacyjnej wskazanym przez *Driver* znajduje się program sterownika graficznego. Utworzenie danej typu *integer* o wartości równej wewnętrznemu numerowi zgłoszonego sterownika.

Uwagi. Niepomyślne wykonanie funkcji *RegisterBGIDriver* powoduje, że rezultatem funkcji *GraphResult* jest dana o wartości

grInvalidDriver — niewłaściwy sterownik graficzny {−4}

RegisterBGIFont (*Graph*)

RegisterBGIFont — zgłoszenie kroju czcionek
(por. *RegisterBGIDriver*)

function *RegisterBGIFont*(*Font* : *pointer*) : *integer*

Zgłoszenie, że w obszarze pamięci operacyjnej wskazanym przez Font znajduje się definicja kroju czcionek. Utworzenie danej typu *integer* o wartości równej wewnętrznemu numerowi tego kroju.

Uwagi. Niepomyślne wykonanie funkcji *RegisterBGIFont* powoduje, że rezultatem funkcji *GraphResult* jest dana o jednej z następujących wartości

grError — nadmierna liczba zgłoszonych krojów czcionek {−11}

grInvalidFont — niewłaściwy krój czcionek {−13}
grInvalidFontNum — niewłaściwy numer kroju czcionek {−14}

RestoreCrtMode (*Graph*)

RestoreCrtMode — przywrócenie trybu tekstowego (por. *DetectGraph*, *InitGraph*, *SetGraphMode*)
procedure *RestoreCrtMode*

Przywrócenie trybu tekstowego, obowiązującego przed ustanowieniem trybu graficznego.

Uwagi. System musi być w trybie graficznym. Użycie procedury *RestoreCrtMode* na przemian z procedurą *SetGraphMode* umożliwia dogodne przełączanie systemu między trybem tekstowym i graficznym.

SetActivePage (*Graph*)

SetActivePage — wybranie strony aktywnej (por. *SetVisualPage*)

procedure *SetActivePage*(*Page* : *word*)

Wybranie strony o numerze *Page* jako strony aktywnej

Uwagi. System musi być w trybie graficznym. Po wykonaniu procedury *SetActivePage* wszystkie operacje graficzne będą dotyczyć strony o podanym numerze. Możliwość wyboru strony aktywnej dotyczy jedynie kart EGA, VGA i Hercules.

SetAllPalette (*Graph*)

SetAllPalette — zmiana kolorów przypisanych pozycjom palety (por. *GetBkColor*, *GetColor*, *GetPalette*, *SetBkColor*, *SetColor*, *SetPalette*)

procedure *SetAllPalette*(**var** *Palette* : *PaletteType*)

Przypisanie identyfikatorów kolorów określonych przez pola rekordu *Palette* typu *PaletteType* pozycjom palety.

Uwagi. System musi być w trybie graficznym. Typ *PaletteType* jest zdefiniowany następująco

```
type
  PaletteType = record
                  Size : byte;
                  Colors : array[0..15] of shortint
                end;
```

Pole *Size* określa rozmiar palety, a pole *Colors* określa identyfikatory kolorów. Jeśli element tablicy *Colors* ma wartość −1, to kolor przypisany odpowiedniej pozycji palety nie ulega zmianie. Identyfikatory kolorów mogą być wyrażone za pomocą następujących symboli

Black — czarny {0}
Blue — niebieski {1}
Green — zielony {2}
Cyan — turkusowy {3}
Red — czerwony {4}
Magenta — karmazynowy {5}
Brown — brązowy {6}
LightGray — jasnoszary {7}
DarkGray — ciemnoszary {8}
LightBlue — jasnoniebieski {9}
LightGreen — jasnozielony {10}
LightCyan — jasnoturkusowy {11}
LightRed — jasnoczerwony {12}
LightMagenta — jasnokarmazynowy {13}
Yellow — żółty {14}
White — biały {15}

SetBkColor (*Graph*)

SetBkColor — zmiana koloru tła
(por. *GetBkColor, GetColor, GetPalette, SetAllPalette, SetColor, SetPalette*)

procedure *SetBkColor(Color : word)*

Zmiana koloru tła na taki kolor, jaki jest związany z pozycją o numerze *Color* bieżącej palety. Wyjątek: jeśli *Color* = 0, to kolor tła zostanie zmieniony na czarny.

Uwagi. System musi być w trybie graficznym.

SetColor (*Graph*)

SetColor — wybranie numeru koloru do wykreślania linii
(por. *GetBkColor, GetColor, GetPalette, SetAllPalette, SetBkColor, SetPalette*)

procedure *SetColor(Color : word)*

Wybranie do wykreślenia linii koloru o numerze *Color*.

Uwagi. System musi być w trybie graficznym.

SetFillPattern (*Graph*)

SetFillPattern – ustalenie wzoru wypełniania obszarów (por. *GetBkColor*, *GetColor*, *GetPalette*) *SetBkColor*, *SetColor*, *SetPalette*)

procedure *SetFillPattern*(*Pattern* : *FillPatternType*;
Color : *word*)

Ustalenie wzoru *Pattern* do wypełniania obszarów (*FillPoly*, *Bar*, *FloodFill*, *Bar3D*, *PieSlice*) i numeru koloru *Color* wzoru wypełniającego.

Uwagi. System musi być w trybie graficznym. Wzór wypełniający jest bitowym prostokątem 8 × 8 pikseli, zapamiętanym w tablicy typu *FillPatternType*. Typ *FillPatternType* jest zdefiniowany następująco

type
FillPatternType = **array**[1..8] **of** *byte*;

SetFillStyle (*Graph*)

SetFillStyle – ustalenie standardowego wzoru wypełniania obszarów (por. *Bar*, *Bar3D*, *GetFillSettings*, *FillPoly*, *PieSlice*)

procedure *SetFillStyle*(*Pattern* : *word*;
Color : *word*)

Ustalenie numeru *Pattern* wzoru do wypełniania obszarów (*FillPoly*, *FloodFill*, *Bar*, *Bar3D*, *PieSlice*) i numeru koloru *Color* wzoru wypełniającego.

Uwagi. System musi być w trybie graficznym. Numery wzorów wypełniających mogą być wyrażone za pomocą następujących symboli:

EmptyFill – wypełnienie kolorem tła {0}
SolidFill – wypełnienie ciągłe {1}
LineFill – wypełnienie pogrubionymi liniami poziomymi {2}
LtSlashFill – wypełnienie liniami pochyłymi {3}
SlashFill – wypełnienie pogrubionymi liniami pochyłymi {4}
BkSlashFill – wypełnienie pogrubionymi liniami ukośnymi {5}
LtBkSlashFill – wypełnienie liniami ukośnymi {6}
HatchFill – wypełnienie siatką pionową {7}
xHatchFill – wypełnienie siatką ukośną {8}
InterleaveFill – wypełnienie liniami splecionymi {9}
WideDotFill – wypełnienie kropkami {10}
CloseDotFill – wypełnienie zagęszczonymi kropkami {11}

SetGraphBufSize (*Graph*)

SetGraphBufSize — ustalenie rozmiaru bufora wykorzystywanego do wypełniania obszarów (por. *InitGraph*)

procedure *SetGraphBufSize*(*BufSize* : *word*)

Ustalenie rozmiaru bufora wykorzystywanego do wypełniania obszarów na wartość *BufSize* bajtów.

Uwagi. System musi być w trybie graficznym. Rozmiar bufora musi być ustalony przed wywołaniem procedury *InitGraph*. Miejsce dla bufora jest przydzielane na stercie.

SetGraphMode (*Graph*)

SetGraphMode — przełączenie systemu do trybu graficznego (por. *ClearDevice*, *DetectGraph*, *GetGraphMode*, *InitGraph*, *RestoreCrtMode*)

procedure *SetGraphMode*(*Mode* : *integer*)

Przełączenie systemu do trybu graficznego o numerze *Mode*, a następnie niejawne wykonanie procedury *GraphDefaults*.

CGAC0	— 320×200, paleta 0 {0}
CGAC1	— 320×200, paleta 1 {1}
CGAC2	— 320×200, paleta 2 {2}
CGAC3	— 320×200, paleta 3 {3}
CGAHi	— 320×200, 1 strona {4}
MCGAC0	— 320×200, paleta 0 {0}
MCGAC1	— 320×200, paleta 1 {1}
MCGAC2	— 320×200, paleta 2 {2}
MCGAC3	— 320×200, paleta 3 {3}
MCGAMed	— 640×200, 1 strona {4}
MCGAHi	— 640×480, 1 strona {5}
EGALo	— 640×200, 16 kolorów, 4 strony {0}
EGAHi	— 640×350, 16 kolorów, 2 strony {1}
EGA64Lo	— 640×200, 16 kolorów, 1 strona {0}
EGA64Hi	— 640×350, 4 kolory, 1 strona {1}
EGAMonoHi	— 640×350,
	64K na karcie, 1 strony {3}
	256K na karcie, 2 strony {3}
HercMonoHi	— 720×348, 2 strony {0}
ATT400C0	— 320×200, paleta 0 {0}
ATT400C1	— 320×200, paleta 1 {1}

ATT400C2 — 320×200, paleta 2 {2}
ATT400C3 — 320×200, paleta 3 {3}
ATT400Med — 640×200, 1 strona {4}
ATT400Hi — 640×400, 1 strona {5}
VGALo — 640×200, 16 kolorów, 4 strony {0}
VGAMed — 640×350, 16 kolorów, 2 strony {1}
VGAHi — 640×480, 16 kolorów, 1 strona {2}
PC3270Hi — 720×350, 1 strona {0}
IBM8514Lo — 640×480, 256 kolorów
IBM8514Hi — 1024×768, 256 kolorów

SetLineStyle (*Graph*)

SetLineStyle — wybranie postaci linii
(por. *GetLineSettings, Line, LineRel, LineTo*)

procedure *SetLineStyle*(*LineStyle* : *word*;
Pattern : *word*;
Thickness : *word*)

Wybranie postaci linii na podstawie *LineStyle* i grubości linii na podstawie *Thickness*. W przypadku gdy *LineStyle* = *UserBitLn*, zdefiniowanie postaci linii na podstawie układu bitów *Pattern*.

Uwagi. System musi być w trybie graficznym. Numery postaci linii mogą być wyrażone za pomocą następujących symboli

SolidLn — linia ciągła {0}
DottedLn — linia kropkowa {1}
CenterLn — linia centrowana {2}
DashedLn — linia przerywana {3}
UserBitLn — linia zdefiniowana {4}

Grubość linii może być wyrażona za pomocą następujących symboli

NormWidth — linia cienka {1}
ThickWidth — linia pogrubiona {3}

SetPalette (*Graph*)

SetPalette — zmiana koloru przypisanego pozycji palety
(por. *GetBkColor, GetColor, GetPalette, SetBkColor, SetColor*)

procedure *SetPalette*(*ColorNum* : *word*;
Color : *byte*)

Przypisanie pozycji *ColorNum* palety, identyfikatora koloru Color.

Uwagi. System musi być w trybie graficznym. Zmiana koloru ujawnia się na ekranie natychmiast po wykonaniu procedury *SetPalette*. Identyfikatory kolorów mogą być wyrażone za pomocą następujących symboli

Black	— czarny {0}
Blue	— niebieski {1}
Green	— zielony {2}
Cyan	— turkusowy {3}
Red	— czerwony {4}
Magenta	— karmazynowy {5}
Brown	— brązowy {6}
LightGray	— jasnoszary {7}
DarkGray	— ciemnoszary {8}
LightBlue	— jasnoniebieski {9}
LightGreen	— jasnozielony {10}
LightCyan	— jasnoturkusowy {11}
LightRed	— jasnoczerwony {12}
LightMagenta	— jasnokarmazynowy {13}
Yellow	— żółty {14}
White	— biały {15}

SetTextJustify (*Graph*)

SetTextJustify — ustalenie sposobu wyrównywania tekstów graficznych
(por. *GetTextSettings*, *OutText*, *OutTextXY*, *SetLineStyle*, *TextHeight*, *TextWidth*)

procedure *SetTextJustify*(*Horiz*: *word*;
Vert : *word*)

Ustalenie sposobu wyrównywania tekstów graficznych w poziomie na podstawie *Horiz* i ustalenie soosobu ich wyrównywania w pionie na podstawie *Vert*.

Uwagi. System musi być w trybie graficznym. Wyrównywanie odbywa się względem punktu wyróżnionego przez kursor graficzny (procedura *OutText*) albo względem punktu o podanych współrzędnych (procedura *OutTextXY*). Wyrównanie w poziomie może być wyrażone za pomocą następujących symboli

LeftText	— wyrównanie do lewej {0}
CenterText	— wyrównanie centryczne {1}
RightText	— wyrównanie do prawej {2}

Wyrównanie w pionie może być wyrażone za pomocą następujących symboli

BottomText – wyrównanie do dołu {0}
CenterText – wyrównanie centryczne {1}
TopText – wyrównanie do góry {2}

SetTextStyle (*Graph*)

SetTextStyle – wybranie kroju czcionek, kierunku wykreślania znaków tekstu i ich rozmiaru (por. *GetTextSettings*, *OutText*, *OutTextXY*, *SetTextJustify*, *TextHeight*, *TextWidth*)

procedure *SetTextStyle*(*Font* : *word*;
Direction : *word*;
CharSize : *word*)

Wybranie kroju czcionek *Font*, kierunku wyprowadzenia *Direction* i rozmiaru czcionek *CharSize*.

Uwagi. System musi być w trybie graficznym. Krój czcionek może być wyrażony za pomocą następujących symboli

DefaultFont – krój domniemany, określony przez wzorzec 8 × 8 bitów {0}
TriplexFont – krój kreskowy potrójny {1}
SmallFont – krój kreskowy indeksowy {2}
SansSerifFont – krój kreskowy bezszeryfowy {3}
GothicFont – krój kreskowy gotycki {4}

Kroje kreskowe charakteryzują się tym, że ich powiększenie nie pogarsza wyglądu czcionki. Kierunek wyprowadzania może być wyrażony za pomocą następujących symboli

HorizDir – poziomo, od lewej do prawej {0}
VertDir – pionowo, od dołu do góry {1}

Czcionki o krojach kreskowych mogą być rozciągane w pionie i w poziomie (por. procedura *SetUserCharSize*). Wymaga to użycia parametru *Size* o wartości określonej przez symbol

UserCharSize – rozmiar programowany {0}

Ponieważ kroje czcionek są zazwyczaj ładowane z dysku, niepomyślne wykonanie procedury *SetTextStyle* powoduje, że rezultatem funkcji *GraphResult* jest dana o jednej z następujących wartości

grFileNotFound – brak zbioru definiującego krój {–8}
grNoFontMem – brak pamięci operacyjnej do zapamiętania kroju {–9}

SetUserCharSize (*Graph*)

SetUserCharSize – zdefiniowanie współczynników rozciągania czcionek krojów kreskowych (por. *SetTextStyle*)

procedure *SetUserCharSize*(*xMul,xDiv* : *byte*;
yMul,yDiv : *byte*)

Zdefiniowanie współczynnika rozciągania w poziomie jako *xMul/xDiv* i współczynnika rozciągania w pionie jako *yMul/yDiv*.

Uwagi. System musi być w tybie graficznym. Po wykonaniu procedury *SetUserCharSize* należy wywołać procedurę *SetTextStyle* z trzecim argumentem o wartości *UserCharSize* {0}.

SetViewPort (*Graph*)

SetViewPort – zdefiniowanie okienka graficznego (por. *ClearViewPort*, *GetViewSettings*, *Window*)

procedure *SetViewPort*(*x1,y1* : *integer*;
x2,y2 : *integer*;
Clip : *boolean*)

Zdefiniowanie okienka graficznego jako prostokąta o współrzędnych przeciwległych wierzchołków (*x1,y1*) i (*x2,y2*). Określenie na podstawie *Clip*, czy wykresy wykraczające poza okienko mają być obcinane.

Uwagi. System musi być w trybie graficznym. Sposób traktowania wykresów wykraczających poza okienko może być wyrażony za pomocą następujących symboli

ClipOff – nie obcinaj {*false*}
ClipOn – obcinaj {*true*}

Po utworzeniu okienka graficznego wszystkie operacje graficzne (z wyjątkiem *SetViewPort*) są wykonywane w tym okienku. Przez domniemanie przyjmuje się, że takim okienkiem jest cały ekran.

SetVisualPage (*Graph*)

SetVisualPage – wybranie strony wyświetlanej (por. *SetActivePage*)

procedure *SetVisualPage*(*Page* : *word*)

Wybranie do wyświetlenia strony o numerze *Page*.

Uwagi. System musi być w trybie graficznym. Możliwość wyboru strony wyświetlanej dotyczy jedynie kart EGA, VGA i Hercules.

TextBackground (*Ctr*)

TexBackground — ustalenie koloru tła
(por. *HighVideo*, *LowVideo*, *NormVideo*, *TextColor*)

procedure *TextBackground*(*Color* : *byte*)

Przypisanie zmiennej *TextAttr* atrybutu koloru tła *Color*.

Uwagi. Atrybut koloru tła jest trzybitowy i może być wyrażony za pomocą następujących symboli:

Black — czarny {0}
Blue — niebieski {1}
Green — zielony {2}
Cyan — turkusowy {3}
Red — czerwony {4}
Magenta — karmazynowy {5}
Brown — brązowy {6}
LightGray — jasnoszary {7}

Po ustaleniu koloru tła, wszystkie znaki wyprowadzane na ekran (włącznie ze spacjami) będą przedstawione na podanym tle.

TextColor (*Crt*)

TextColor — ustalenie koloru pierwszego planu
(por. *HighVideo*, *LowVideo*, *NormVideo*, *TextBackground*)

procedure *TextColor*(*Color* : *byte*)

Przypisanie zmiennej *TextAttr* atrybutu koloru znaku (i migotania) o wartości *Color*.

Uwagi. Atrybut koloru znaku jest czterobitowy i może być określony za pomocą następujących symboli:

Black — czarny {0}
Blue — niebieski {1}
Green — zielony {2}
Cyan — turkusowy {3}
Red — czerwony {4}
Magenta — karmazynowy {5}
Brown — brązowy {6}
LightGray — jasnoszary {7}
DarkGray — ciemnoszary {8}
LightBlue — jasnoniebieski {9}
LightGreen — jasnozielony {10}

LightCyan — jasnoturkusowy {11}
LightRed — jasnoczerwony {12}
LightMagenta — jasnokarmazynowy {13}
Yellow — żółty {14}
White — biały {15}
Blink — migotanie {128}

Po ustaleniu koloru znaku, wszystkie znaki będą wprowadzane na ekran w tym kolorze.

TextHeight (*Graph*)

TextHeight — wyznaczenie wysokości tekstu (por. *OutText*, *OutTextXY*, *SetTextStyle*, *SetUserCharSize*, *TextWidth*)

function *TextHeight*(*TextString* : *string*) : *word*

Utworzenie danej typu *word* o wartości równej wyrażonemu w pikselach pionowemu rozmiarowi tekstu *TextString*.

Uwagi. System musi być w trybie graficznym. Rezultat uwzględnia współczynnik rozciągania znaków w pionie.

TextMode (*Crt*)

TextMode — ustanowienie trybu tekstowego (por. *RestoreCrt*)

procedure *TextMode*(*Mode* : *word*)

Zmiana bieżącego trybu tekstowego na tryb tekstowy *Mode*.

Uwagi. Nowy tryb tekstowy może być wyrażony za pomocą następujących symboli:

BW40 — 40×80, czarno-biały {0}
C40 — 40×25, kolorowy {1}
BW80 — 80×25, czarno-biały {2}
C80 — 80×25, kolorowy {3}
Mono — 80×25, czarno-biały (na monitorze monochromatycznym) {7}

Dodatkową możliwością określenia trybu jest odwołanie się do zmiennej *LastMode*. W prologu zmiennej tej jest przypisywany numer trybu obowiązującego przed podjęciem wykonywania programu. W celu umożliwienia wyprowadzania na ekran 43 wierszy tekstu (karta EGA) albo 50 wierszy (karta VGA), zdefiniowano dodatkowy symbol

Font8x8 {$100}

W szczególności wywołanie

TextMode(CO80 + Font8x8)

powoduje ustanowienie trybu tekstowego kolorowego 80 × 25 z wyprowadzaniem 43 albo 50 wierszy. Pozostaje nadmienić, że bezpośrednio po wywołaniu procedury *TextMode*, bieżącym okienkiem tekstowym staje się cały ekran, okienko zostaje wyczyszczone, a zmiennej *TextAttr* zostają przypisane atrybuty jak po wywołaniu procedury *NormVideo*.

TextWidth (*Graph*)

TextWidth — wyznaczenie szerokości tekstu (por. *OutText*, *OutTextXY*, *SetTextStyle*, *SetUserCharSize*, *TextHeight*)

function *TextWidth(TextString : string) : word*

Utworzenie danej typu *word* o wartości równej wyrażonemu w pikselach poziomemu rozmiarowi tekstu *TextString*.

Uwagi. System musi być w trybie graficznym. Rezultat uwzględnia współczynnik rozciągania znaków w poziomie.

WhereX (*Crt*)

WhereX — udostępnienie poziomej współrzędnej kursora tekstowego (por. *GotoXY*, *WhereY*, *Window*)

function *WhereX : byte*

Utworzenie danej typu *byte* o wartości równej poziomej współrzędnej kursora tekstowego.

Uwagi. Współrzędna dotyczy bieżącego okienka tekstowego. Lewy górny znak okienka ma współrzędną poziomą 1.

WhereY (*Crt*)

WhereY — udostępnienie pionowej współrzędnej kursora tekstowego (por. *GotoXY*, *WhereX*, *Window*)

function *WhereY : byte*

Utworzenie danej typu *byte* o wartości równej pionowej współrzędnej kursora tekstowego.

Uwagi. Współrzędna dotyczy bieżącego okienka tekstowego. Lewy górny znak okienka ma współrzędną pionową 1.

Window (*Crt*)

Window – utworzenie okienka tekstowego (por. *SetViewPort*)

procedure *Window*(*x1*,*y1* : *byte*;
x2,*y2* : *byte*)

Zdefiniowanie okienka tekstowego jako prostokąta, którego przeciwległe wierzchołki mają współrzędne (*x1*,*y1*) i (*x2*,*y2*).

Uwagi. Współrzędne są liczone względem ekranu. Lewy górny narożnik ekranu ma współrzędne (1,1). Najmniejsze okienko składa się z jednej kolumny i jednego wiersza. Okienko domniemane ma współrzędne (1,1,80,25) w trybach 80-kolumnowych i współrzędne (1,1,40,25) w trybach 40-kolumnowych. W przypadku błędnego określenia współrzędnych okienka, wykonanie procedury *Window* nie wywołuje żadnych skutków. Po ustanowieniu okienka tekstowego, wszystkie współrzędne tekstowe (z wyjątkiem wymienionych w procedurze *Window*) są liczone względem okienka. Współrzędne ekranowe okienka (każda zmniejszona o 1) są zapamiętywane w zmiennych *WindMin* i *WindMax*. Pierwsza z nich określa współrzędne lewego górnego, a druga prawego dolnego narożnika okienka. Pierwszy bajt każdej z tych zmiennych zawiera współrzędną *x*, a drugi współrzędną *y*.

Wybrane definicje i deklaracje modułów *Crt* i *Graph*

(opracowano za zgodą Borland International na podstawie zbiorów Crt.doc i Graph.doc znajdujących się na dyskietce dystrybucyjnej systemu Turbo Pascal 4.0)

Turbo Pascal Version 4.0
CRT Unit Interface Documentation

Copyright (c) 1987 Borland International, Inc.

```
unit Crt;
interface
const
{ CRT modes }
  BW40      = 0;      { 40×25 B/W on Color Adapter }
  CO40      = 1;      { 40×25 Color on Color Adapter }
  BW80      = 3;      { 80×25 B/W on Color Adapter }
```

```
   CO80         = 3;      { 80×25 Color on Color Adapter }
   Mono         = 7;      { 80×25 on Monochrome Adapter }
   Font8x8      = 256;    { Add-in for ROM font }

   { Foreground and background color }
   Black        = 0;
   Blue         = 1;
   Green        = 2;
   Cyan         = 3;
   Red          = 4;
   Magenta      = 5;
   Brown        = 6;
   LightGray    = 7;

   { Foreground color }
   DarkGray     = 8;
   LightBlue    = 9;
   LightGreen   = 10;
   LightCyan    = 11;
   LightRed     = 12;
   LightMagenta = 13;
   Yellow       = 14;
   White        = 15;

   { Add-in for blinking }
   Blink        = 128;
var
   LastMode: word;          { Current text mode }
   TextAttr: byte;          { Current text attribute }
   WindMin: word;           { Window upper left coordinates }
   WindMax: word;           { Window lower right coordinates }

procedure TextMode(Mode: word);
procedure Window(x1,y1,x2,y2: byte);
function WhereX: byte;
function WhereY: byte;
procedure ClrScr;
procedure ClrEol;
procedure InsLine;
function KeyPressed : boolean;
procedure DelLine;
procedure TextColor(Color: byte);
procedure TextBackground(Color: byte);
procedure TextMode(Mode : word);
```

```
procedure LowVideo;
procedure HighVideo;
procedure NormVideo;
```

Turbo Pascal Version 4.0 *GRAPH Unit Interface Documentation* *Copyright (c) 1987 by Borland International, Inc.*

```
unit Graph;
interface
const
  { GraphResult error return codes }
  grOK               =   0;
  grNoInitGraph      =  -1;
  grNotDetected      =  -2;
  grFileNotFound     =  -3;
  grInvalidDriver    =  -4;
  grNoLoadMem        =  -5;
  grNoScanMem        =  -6;
  grNoFloodMem       =  -7;
  grFontNotFound     =  -8;
  grNoFontMem        =  -9;
  grInvalidMode      = -10;
  grError            = -11;    { Generic error }
  grIOerror          = -12;
  grInvalidFont      = -13;
  grInvalidFontNum   = -14;
  grInvalidDeviceNum = -15;

  { Graphics drivers }
  Detect   = 0;   { Requests autodetection }
  CGA      = 1;
  MCGA     = 2;
  EGA      = 3;
  EGA64    = 4;
  EGAMono  = 5;
  IBM8514  = 6;   { RESERVED }
  HercMono = 7;
  ATT400   = 8;
  VGA      = 9;
  PC3270   = 10;
```

```
{ Graphics modes for each driver }
CGAC0      = 0;  { 320×200 palette 0:
                   LightGreen, LightRed, Yellow;
                   1 page }
CGAC1      = 1;  { 320×200 palette 1:
                   LightCyan, LightMagenta, White;
                   1 page }
CGAC2      = 2;  { 320×200 palette 2:
                   Green, Red, Brown;
                   1 page }
CGAC3      = 3;  { 320×200 palette 3:
                   Cyan, Magenta, LightGray;
                   1 page }
CGAHi      = 4;  { 640×200 1 page }
MCGAC0     = 0;  { 320×200 palette 0:
                   LightGreen, LightRed, Yellow;
                   1 page }
MCGAC1     = 1;  { 320×200 palette 1:
                   LightCyan, LightMagenta, White;
                   1 page }
MCGAC2     = 2;  { 320×200 palette 2:
                   Green, Red, Brown;
                   1 page }
MCGAC3     = 3;  { 320×200 palette 3:
                   Cyan, Magenta, LightGray;
                   1 page }
MCGAMed    = 4;  { 640×200 1 page }
MCGAHi     = 5;  { 640×480 1 page }
EGALo      = 0;  { 640×200 16 color 4 page }
EGAHi      = 1;  { 640×350 16 color 2 page }
EGA64Lo    = 0;  { 640×200 16 color 1 page }
EGA64Hi    = 1;  { 640×350 4 color 1 page }
EGAMonoHi  = 3;  { 640×350 64K on card, 1 page;
                           256K on card, 2 page }
HercMonoHi = 0;  { 720×348 2 page }
ATT400C0   = 0;  { 320×200 palette 0:
                   LightGreen, LightRed, Yellow;
                   1 page }
ATT400C1   = 1;  { 320×200 palette 1:
                   LightCyan, LightMagenta, White;
                   1 page }
```

```
ATT400C2    = 2;  { 320×200 palette 2:
                    Green, Red, Brown;
                    1 page }
ATT400C3    = 3;  { 320×200 palette 3:
                    Cyan, Magenta, LightGray;
                    1 page }
ATT400Med   = 4;  { 640×200 1 page }
ATT400Hi    = 5;  { 640×400 1 page }
VGALo       = 0;  { 640×200 16 color 4 page }
VGAMed      = 1;  { 640×350 16 color 2 page }
VGAHi       = 2;  { 640×480 16 color 1page }
PC3270Hi    = 0;  { 720×350 1 page }
IBM8514Lo   = 0;  { 640×480, 256 color } { RESERVED }
IBM8514Hi   = 1;  { 1024×768, 256 color } { RESERVED }
{ Colors for SetPalette and SetAllPalette }
Black        = 0;
Blue         = 1;
Green        = 2;
Cyan         = 3;
Red          = 4;
Magenta      = 5;
Brown        = 6;
LightGray    = 7;
DarkGray     = 8;
LightBlue    = 9;
LightGreen   = 10;
LightCyan    = 11;
LightRed     = 12;
LightMagenta = 13;
Yellow       = 14;
White        = 15;

{ Line style and widths for Get/SetLineStyle }
SolidLn    = 0;
DottedLn   = 1;
CenterLn   = 2;
DashedLn   = 3;
UserBitLn  = 4;  { User-defined line style }
NormWidth  = 1;
ThickWidth = 3;

{ Set/GetTextStyle }
DefaultFont  = 0;  { 8×8 bit mapped font }
```

```
TriplexFont   = 1;  { Stroked fonts }
SmallFont     = 2;
SansSerifFont = 3;
GothicFont    = 4;

HorizDir = 0;       { Left to right }
VertDir  = 1;       { Bottom to top }

UserCharSize = 0;   { User-defined char size }

{ Clipping symbols }
ClipOn = true;
ClipOff = false;

{ Bar3D symbols }
TopOn  = true;
TopOff = false;

{ Fill patterns for Get/SetFillStyle }
EmptyFill      = 0;   { fills area in background color }
SolidFill      = 1;   { fills area in solid fill color }
LineFill       = 2;   { --- fill }
LtSlashFill    = 3;   { /// fill }
SlashFill      = 4;   { /// fill with thick lines }
BkSlashFill    = 5;   { \ fill with thick lines }
LtBkSlashFill  = 6;   { \ fill }
HatchFill      = 7;   { light hatch fill }
XHatchFill     = 8;   { heavy cross hatch fill }
InterleaveFill = 9;   { interleaving line fill }
WideDotFill    = 10;  { widely spaced dot fill }
CloseDotFill   = 11;  { closely spaced dot fill }
UserFill       = 12;  { user defined fill }

{ BitBlt operators for PutImage }
NormalPut = 0;  { MOV }
xorPut    = 1;  { XOR }
orPut     = 2;  { OR }
andPut    = 3;  { AND }
notPut    = 4;  { NOT }

{ Horizontal and vertical justification
                     for SetTextJustify }
LeftText   = 0;
CenterText = 1;
RightText  = 2;

BottomText= 0;
```

```
{ CenterText = 1; already defined above }
  TopText    = 2;
const
  MaxColors = 15;
type
  PaletteType = record
                  Size   : byte;
                  Colors : array[0..MaxColors] of shortint
                end;
  LineSettingsType = record
                       LineStyle : word;
                       Pattern   : word;
                       Thickness : word
                     end;
  TextSettingsType = record
                       Font      : word;
                       Direction : word;
                       CharSize  : word;
                       Horiz     : word;
                       Vert      : word
                     end;
  { Pre-defined fill style }
  FillSettingsType = record
                       Pattern : word;
                       Color   : word
                     end;
  { User defined fill style }
  FillPatternType = array[1..8] of byte;
  PointType = record
                x, y: integer
              end;
  ViewPortType = record
                   x1, y1, x2, y2 : integer;
                   Clip           : boolean
                 end;
  ArcCoordsType = record
                    x,y            : integer;
                    xStart, yStart : integer;
                    xEnd, yEnd     : integer
                  end;
{ *** High-level error handling *** }
```

```
function GraphErrorMsg(ErrorCode : integer) : string;
function GraphResult : integer;
{ *** Detection, initialization and crt mode routines  *** }
procedure DetectGraph (var GraphDriver,
                           GraphMode : integer);
procedure InitGraph (var GraphDriver : integer;
                     var GraphMode : integer;
                         PathToDriver : string);
function RegisterBGIfont(Font : pointer) : integer;
function RegisterBGIdriver(Driver : pointer) : integer;
procedure SetGraphBufSize(BufSize : word);
procedure GetModeRange(GraphDriver : integer;
                       var LoMode, HiMode : integer);
procedure SetGraphMode(Mode : integer);
function GetGraphMode : integer;
procedure GraphDefaults;
procedure RestoreCrtMode;
procedure CloseGraph;
function GetX : integer;
function GetY : integer;
function GetMaxX : integer;
function GetMaxY : integer;
{ *** Screen, viewport, page routines ***}
procedure ClearDevice;
procedure SetViewPort(x1, y1, x2, y2 : integer;
                      Clip : boolean);
procedure GetViewSettings(var ViewPort : ViewPortType);
procedure ClearViewPort;
procedure SetVisualPage(Page : word);
procedure SetActivePage(Page : word);
{ *** Point-oriented routines *** }
procedure PutPixel (x, y : integer; Pixel : word);
function GetPixel (x, y : integer) : word;
{ *** Line-oriented routines *** }
procedure LineTo (x, y : integer);
procedure LineRel (dx, dy : integer);
procedure MoveTo (x, y : integer);
procedure MoveRel (dx, dy : integer);
procedure Line (x1, y1, x2, y2 : integer);
procedure GetLineSettings (var LineInfo : LineSettingsType);
procedure SetLineStyle (LineStyle : word;
```

```
                         Pattern   : word;
                         Thickness : word);
{ *** Polygon, fills and figures *** }
procedure Rectangle (x1, y1, x2, y2 : integer);
procedure Bar (x1, y1, x2, y2 : integer);
procedure Bar3D (x1, y1, x2, y2 : integer);
                 Depth : word; TopFlag : boolean);
procedure DrawPoly (NumPoints : word; var PolyPoints);
procedure FillPoly (NumPoints : word; var PolyPoints);
procedure GetFillSettings (var FillInfo : FillSettingsType);
procedure GetFillPattern (var FillPattern : FillPatternsType);
procedure SetFillStyle (Pattern : word; Color : word);
procedure SetFillPattern (Pattern : FillPatternType;
                          Color : word);
procedure FloodFill (x, y : integer; Border : word);
{ *** Arc, circle, and other curves *** }
procedure Arc (x, y : integer; StAngle, EndAngle,
               Radius : word);
procedure GetArcCoords(var ArcCoords : ArcCoordsType);
procedure Circle (x, y : integer; Radius : word);
procedure Ellipse (x, y : integer;
                   StAngle, EndAngle : word;
                   xRadius, yRadius : word);
procedure GetAspectRatio (var xAsp, yAsp : word);
procedure PieSlice (x, y : integer; StAngle, EndAngle,
                    Radius : word);
{ *** Color and palette routines *** }
procedure SetBkColor (Color : word);
procedure SetColor (Color : word);
function GetBkColor : word;
function GetColor : word;
procedure SetAllPalette (var Palette : PaletteType);
procedure SetAllPalette (ColorNum : word; Color : shortint);
procedure GetPalette (var Palette : PaletteType);
function GetMaxColor : word;
{ *** Bit-image routines *** }
function ImageSize (x1, y1, x2, y2 : integer) : word;
procedure GetImage (x1, y1, x2, y2 : integer; var BitMap);
procedure PutImage (x, y : integer; var BitMap;
                    Op : word);
{ *** Text routines *** }
```

procedure *GetTextSettings* (**var** *TextInfo* : *TextSettingsType*);
procedure *OutText* (*TextString* : *string*);
procedure *OutTextXY* (*x*, *y* : *integer*; *TextString* : *string*);
procedure *SetTextJustify* (*Hor*, *Vert* : *word*);
procedure *SetTextStyle* (*Font*, *Direction* : *word*;
CharSize : *word*);
procedure *SetUserCharSize* (*MultX*, *DivX*, *MultY*, *DivY* : *word*);
function *TextHeight* (*TextString* : *string*) : *word*;
function *TextWidth* (*TextString* : *string*) : *word*;

Literatura

1. Cherry G.: *Pascal Programming Structures*. Reston Publishing Company, Reston, Virginia 1980.
2. Iglewski M., Madey J., Matwin S.: *Pascal*, WNT, Warszawa 1986.
3. Jensen K., Wirth N.: *Pascal — User Manual and Report*. Springer Verlag, Berlin 1974.
4. Schauer H.: *Pascal für Anfanger*. R. Oldenbourg Verlag, Wien 1976.
5. Turbo Pascal version 3.0. Borland International. Scotts Valley, California 1985.
6. Turbo Toolbox. Borland International. Scotts Valley, California 1985.
7. Turbo Tutor. Borland International. Scotts Valley, California 1985.
8. Walasek J.: *Konwersacyjne otoczenie programowe Pascala*. WNT, Warszawa 1983.

Skorowidz

WNT Warszawa 1989.
Wydanie I. Nakład 19 700+ 300 egz.
Ark. wyd. 19,3. Ark. druk. 20,25.
Format B5. Papier offset kl. III, 70 g.
Podpisano do druku w lipcu 1989.
Druk ukończono w sierpniu 1989.
Symbol Et/82305/WNT
Szczecińskie Zakłady Graficzne
Zam. 0372/1110/89/II U-61

ISBN 83-204-0944-6